Survival analysis of ICU patients using the Mimic-iv dataset

1 CONTENTS

2 INTRODUCTION

This study is aimed to predict if and when patients, when admitted into the ICU, will die. In recent times Electronic Health Records give a huge amount of data for clinical risk modelling. In the past, clinical models tend to remain within large registries that abstract data, into static snapshots of patient health (McNamara et al., 2016). As, in ICUs the frequency of data collection is on an hourly rate, it has allowed vast medical databases to arise such as MIMIC-IV. This can allow different modelling techniques to be used such as machine learning and Ai. In this work, we will be comparing the classical statistical methodologies such as Cox regression, accompanied by the Kaplan-Meier Curve (Bewick, Cheek, Ball, 2004) to the newer machine learning algorithms, Random survival forests and Gradient Boosted Machine, to find the pros and cons of both methods.

Machine learning methods have shown real promise in predicting important clinical outcomes such as mortality (Lee, Chen and Ishwaran, 2021; Schulz, Kvedar and Krumholz, 2020), however they tend to only produce one classification prediction at one point in time during the time period being analysed. For example, (Mortazavi et al., 2017) uses the first 24 hours of data to predict outcomes after cardiovascular procedures. Other forms of prediction models that are time dependent dynamically update predictions using the latest data given (Ma et al., 2019; Wang et al., 2020). In an ICU this makes the outcomes much more useful as when a patient is in critical care the health status can change very rapidly.

An existing real-time prediction model is the Cox regression model. This is however a classical statistical prediction methodology, thus does not take advantage of the modern strides in machine learning achieved.

The aim of this work is to compare the differences in outcomes of the new survival analysis technique with the more classical but real-time model. To find which should be used in an ICU setting. This is important research as it allows hospital workers to understand which patients are at the highest risk of death and can help prioritize care for people that need it the most, which will help keep as many people alive as possible.

To complete this task, we will be using all the three algorithms to model fatality percentage based on the inputs of vital signatures such as ion blood levels, weight, glucose levels. We will then be reviewing each model and finding out which model is the best at predicting fatality. These health statistics are derived from the MIMIC-IV database. MIMIC-IV is a publicly available dataset provided by the Laboratory for Computational Physiology (LCP) at the Massachusetts Institute of Technology (MIT). This is a large public database on 37000 patients holding all the vital information recorded at admission from 2008 – 2019, from an ICU in Israel. The data is derived from a hospital-wide Electronic Health Record (EHR) and an Intensive Care Unit (ICU). The data is derived from a singular hospital, however in the future, the goal stated from MIT is to incorporate data from multiple institutions capable of supporting research on critically ill patients worldwide.

For my analysis we will be using RStudio and Python. Each software has package implementation which allows us to install independently created commands and functions which allows survival analysis to be performed faster and more efficiently. In RStudio, some of the packages used are ggsurvfit, survminer and mlr, and in Python, some of the packages used are Pandas, Sklearn, and sksurv. These are all very widely used packages in the data science industry and allow the algorithms to be run in just a few lines of code.

For the rest of the report, we will be going through the methodology of each algorithm and explaining how and why we have implemented them in the chosen way. We then compare the hazard score results to see which model had the best performance when predicting fatality in the patients. To find out overall which model is best, we will compare 2 aspects of the performance of each model, them being, concordance index and computational efficiency, to see which model was best overall.

3 METHODS

3.1 PACKAGES

The cox regression and Kaplan-Meier Curves were both created in R, while the machine learning algorithms were created in Python, the packages used in both languages are as follows.

R	Python
Survival	Numpy
ggsurvfit	Matplotlib
gtsummary	Pandas
tidycmprsk	joblib
condSURV	Sklearn
dplyr	Sklearn.model_selection
survminer	Sklearn.preprocessing
tidyverse	Sksurv.ensemble
mlr	Sksurv.preprocessing

3.2 COX REGRESSION

The first of the methods used were the Cox Proportional Hazards Survival Regression model, in conjunction with the Kaplan-Meier Curve. These are the only classical statistical model used which will later be evaluated by comparing them to the Random Survival Forests machine learning algorithm and Gradient Boosted machine learning algorithm.

Cox regression is used in survival analysis to determine the influence of different variables on survival time, by proportional hazards model. These different survival times are measured and then plotted over a time series graph, showing how many patients were alive at certain points in the study. Each patient will have a start time, given by the time entering the ICU and an event time of when they died, the time between these two events, given in hours, is considered in the survival time analysis. When completing survival analysis in simple problem, with only 1 to 2 predictors, a log rank test can be used, however in this case, the predictors are the vital signatures of the patients in the ICU. This means that our models will be using 195 predictors, including age, implying that a simple log rank test will not satisfy a solution for our problem, thus a complete Cox Proportional Hazards Survival Regression model will be used. As the vital variables in mimic-iv are split into 6 categories, amin, amax, kurtosis, mean, skew and standard deviation. Cox regression can be performed on each category of variable, calculating 6 different concordance indexes' allowing us to compare which type of variable had the greatest detrimental effect on survival. Cox regression does however have some prerequisite assumptions made on the data, the first assumption is of linearity, meaning there is a linear relationship between the log hazard and the continuous covariates (explanatory variables). The second assumption is the additive assumption, stating that the influence of a predictor variable on the dependant variable is independent of any other influence.

Finally, the fundamental assumption for cox regression is that the hazards are proportional, meaning that the relative hazard remains constant over time. To note there is no assumptions on the structure of the data, meaning that imputation and scaling is not required to complete the method. (Kuitunen, Ponkilainen, Uimonen, Eskelinen, Reito; 2021), however we will see an improvement in results. Before imputing, a missingness threshold was applied to the dataset of 30%, meaning that if a column had more than 30% of its values missing then instead of imputing that column, it would be dropped instead. This is to keep the accuracy of the database intact, as clearly, any imputed value is not the true value of the study, thus the accuracy of the results would decrease. So, a threshold is set to where if more than that many values must be imputed the variable is dropped from the database entirely. Any variable that didn't meet this threshold (had less than 30% missing) was imputed at a mean level, the mean of that column was the value imputed this allowed the value to have some relation to the dataset and reduce the anomalies created when imputing.

The cox regression model is based upon a hazard function output, which will be used as a descriptor of the survival probability of the patient. The way this is calculated is as follows:

$$H(t) = H_0(t) \times \exp[b_1 x_1 + b_2 x_2 + \cdots .. b_k x_k]. \quad (1)$$

Where x1 to xk are the predictor variables (or vitals), H0(t) is the base line hazard function given at time t, equalling the hazard function when all predictor variables are set to 0. By calculating the exponential of the regression coefficients (b1 to bk) we can calculate the corresponding risk factor of the model. A regression coefficient is defined as the amount by which change in the x must be multiplied to give the corresponding average change in y, or vice versa.

Once cox regression has been completed, a value called the concordance index is given, a concordance index is a measure of performance of the model, by outputting the percentage of correct predictions the model achieved. This can be shown as a either the concordance index or graphically with an AUC ROC curve. As the method is either guessing a binary output, the only answers are either 1 or 0, so if the prediction or randomly guessing you would expect a value of 0.5 as the output, thus for our models to be worthwhile we are looking for a value to be as close as possible to 1. Cox regression also has an importance output associated with the model. This output will rank all the variables in the cox regression and ranks them on impact with the survival probability. This will show us with variables affect survival probability the most.

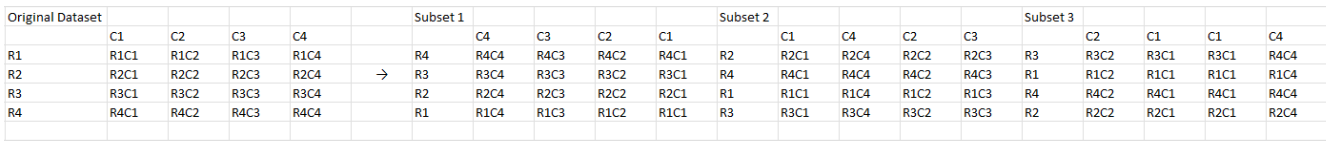
3.3 Kaplan-Meier Curves

As well as Cox regression, Kaplan-Meier curves were produced. The Kaplan-Meier curve is a graphical representation of the survival function, named after Edward Kaplan and Meier, who developed the technique in the 1950s. It is a non-parametric estimate of the survival function that does not make any assumptions about the underlying distribution of the data. The Kaplan-Meier curve is used to estimate the survival function from data that is censored truncated or have missing values. It shows the probability that a subject will survive up to a time t. The plot is constructed by plotting the survival function against time.

To differentiate from the cox regression models described earlier, the Kaplan-Meier curves plotted the dataset as a whole, using all available vital variables, not plotting each category of variables separately. This allowed us to understand how the survival probability of patients changed overtime and allowed us to understand the dataset better as a whole.

3.4 Random Survival Forests

Random Survival Forest is a nonparametric tree-based ensemble method for the analysis of right censored survival data. One could define right censored survival data as when the survival time is only known to exceed to a determined certain value. This method is built as a time-to-event extension of random forests for classification, where it uses a large amount of binary decision trees to recursively partition the covariant space forming groups of subjects with similar survival probability functions (Pickett, Suresh, Campbell, Davis, Juarez-Colunga; 2021). Then complete majority voting to gain an average formation of leaf nodes as the true classification (see figure 1). The different decision trees are formed by bootstrapping the original dataset, we randomly select rows and columns out of the dataset, these selected rows and columns are then placed in to form new datasets of the same rearranged data (see figure 2). The reason why this is an improvement on just a singular binary decision trees is because the trained tree is highly dependent on the original dataset, changing a very small amount of variables, can drastically change the tree; accuracy of correct prediction on the training set would be very strong, but due to the high variance, weak translation to the test set. Random survival forest has no fundamental assumptions to be used meaning that in recent times it has gained prominence as a replacement to cox regression has it does not have the 3 assumptions discussed earlier. (Shah, A. D., Bartlett, J. W., Carpenter, J., Nicholas, O., & Hemingway, H. 2014)



Original Dataset						Subset 1					Subset 2					Subset 3					
	C1	C2	C3	C4			C4	C3	C2	C1		C1	C4	C2	C3		C2	C1	C1	C4
R1	R1C1	R1C2	R1C3	R1C4		R4	R4C4	R4C3	R4C2	R4C1	R2	R2C1	R2C4	R2C2	R2C3	R3	R3C2	R3C1	R3C1	R4C4
R2	R2C1	R2C2	R2C3	R2C4	→	R3	R3C4	R3C3	R3C2	R3C1	R4	R4C1	R4C4	R4C2	R4C3	R1	R1C2	R1C1	R1C1	R1C4
R3	R3C1	R3C2	R3C3	R3C4		R2	R2C4	R2C3	R2C2	R2C1	R1	R1C1	R1C4	R1C2	R1C3	R4	R4C2	R4C1	R4C1	R4C4
R4	R4C1	R4C2	R4C3	R4C4		R1	R1C4	R1C3	R1C2	R1C1	R3	R3C1	R3C4	R3C2	R3C3	R2	R2C2	R2C1	R2C1	R2C4

Figure 1: Simple Bootstrapping example

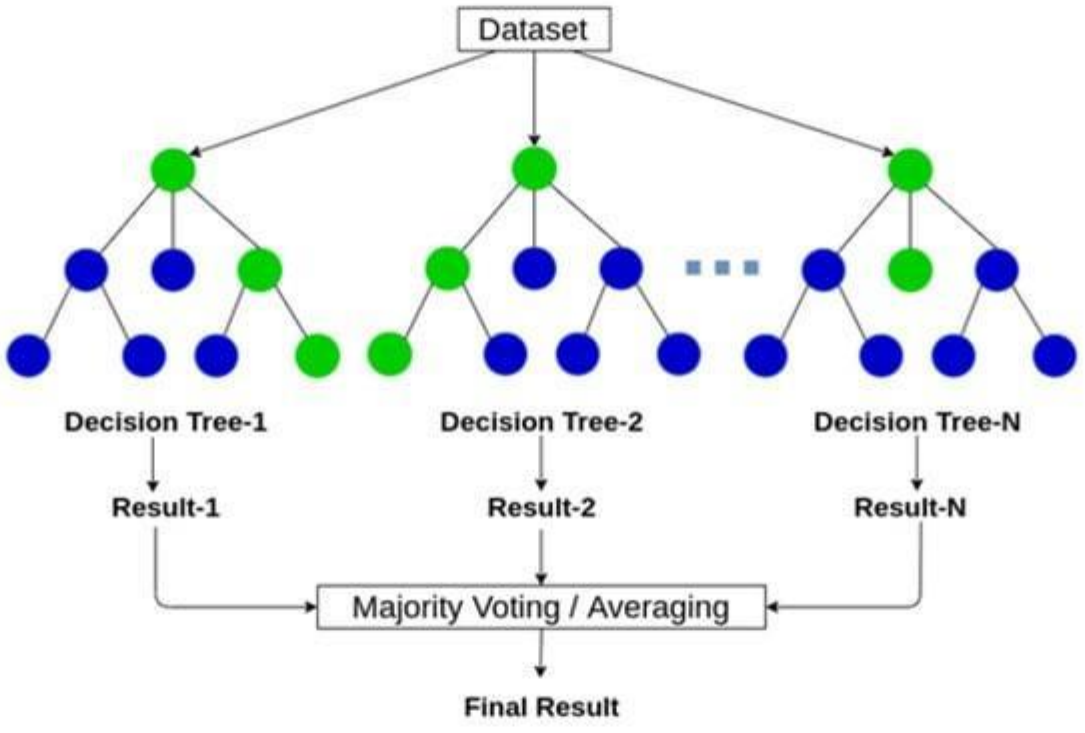


Figure 2: Random Survival Forest Example: Hao, L., Kim, J., Kwon, S., & Ha, I. do. (2021).

From this we can see that the steps taken by the algorithm to complete random survival forest are as follows:

Step 1: Draw B bootstrap values.

Step 2: Grow a survival Tree based upon the data of each bootstrap samples (b = 1,…,B):

a) At each tree node select a subset of the predictor variables.
b) Among all binary splits defined by the predictor variables selected in (a), find the best split into two subsets (the daughter nodes) according to a suitable criterion for the right censored data, like the log rank test.
c) Repeat (a) – (b) recursively on each daughter node until a stopping criterion is met.

Step 3: Aggregate information from the terminal nodes (nodes with no further split) from B survival trees to obtain a risk prediction ensemble.

Mogensen, U. B., & Gerds, T. A. (n.d.).

The bootstrapping sample. Using the counting process notation.

$$\tilde{N}_i(s) = \mathscr{I}\left(\tilde{T}_i \le s, \Delta_i = 1\right); \qquad \tilde{Y}_i(s) = \mathscr{I}\left(\tilde{T}_i > s\right),$$

we have

$$\tilde{N}_b^*(s, \mathbf{x}) = \sum_{i=1}^{N} c_{ib} \mathscr{I}\left(X_i \in \mathscr{T}_b(\mathbf{x})\right) \tilde{N}_i(s); \qquad \tilde{Y}_b^*(s, \mathbf{x}) = \sum_{i=1}^{N} c_{ib} \mathscr{I}\left(X_i \in \mathscr{T}_b(\mathbf{x})\right) \tilde{Y}_i(s).$$

Figure 3: Bootstrapping sample

$$\hat{H}_b(t|\mathbf{x}) = \int_0^t \frac{\tilde{N}_b^*(\mathrm{d}s, \mathbf{x})}{\tilde{Y}_b^*(s, \mathbf{x})}.$$

The ensemble survival function from random survival forest is

$$\hat{S}^{rsf}(t|\mathbf{x}) = exp\left(-\frac{1}{B}\sum_{b=1}^{B} \hat{H}_b(t|\mathbf{x})\right). \qquad (1)$$

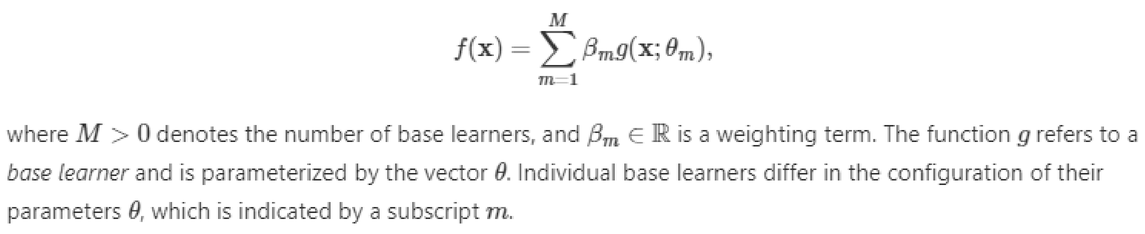
Figure 4: Random Survival Forests Survival Function

Where Nb(s,x) is the bth node in the forest, for every value lower than s and Yb is the bth node above s. The hazard function at b is given by Hb and is calculated by the integral of Nb/Yb, and the survival function is the sum of hazard functions.

Just like in cox regression, we are evaluating the performance of our model with the concordance index. We are also finding the most important variables to see which specific variables have the most impact on survivability.

3.5 Gradient Boosting Machine

Unlike Random Survival Forest, Gradient boosting does not refer to one specific model, instead it refers to a method of optimizing a given loss function, giving more predictions closer to the true value. It follows an iterative process of combining multiple weak leaners to achieve a powerful model. Weak leaners are defined as an underpowered weak leaner that doesn't show all the larger dynamics of the data, in our case when we implemented the general Gradient Boosting sksurv function GradientBoostingSurvivalAnalysis(), we were using regression tree based leaners whereas, when we implemented the ComponentwiseGradientBoostingSurvivalAnalysis(), the base leaners were component wise least squares. The general model is more versatile than component wise as, component wise creates a linear model at the end, but only for a smaller subset of variables used, thus we should expect the general model to have the higher performance. This means that the overall model f(x) is:



$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m g(\mathbf{x}; \theta_m),$$

where $M > 0$ denotes the number of base learners, and $\beta_m \in \mathbb{R}$ is a weighting term. The function g refers to a *base learner* and is parameterized by the vector θ. Individual base learners differ in the configuration of their parameters θ, which is indicated by a subscript m.

Figure 4: Gradient Boosting Machine General Model (scikit-survival)

When completing the gradient boosting algorithm, the loss function of the weak leaners is plotted with a prediction value and $\hat{y}$ and a true value y as an input giving Loss Function = $L(y, \hat{y})$ (*Haihao Lu, Sai Praneeth Karimireddy, Natalia Ponomareva, Vahab Mirrokni(2020)*. We then compute the differential of these predictor values, finding the gradient of the loss function at that predictor calling this matrix θ_m.

$$\theta_m = \frac{dL(y, \hat{y})}{d\,\hat{y}}$$

As there is a distance between the predictor and truth, the gradient will inform the model on which direction the true value is in, if the gradient is positive, we know that the true value is to the left of the predicted, and if the gradient is negative, then the true value is to the right of the predictor. We will then fit a new learner by combing the product these gradients with a weighting function β_m.

$$f_2 = f_1 + \beta_m \vartheta_m$$

We only want to incrementally move in the direction of the gradient, we cannot add combine the whole gradient, as that could lead to under or overshooting the true value. This is why we have added a weighting function, this weighting function is calculated by finding the minimum loss between $f_1 + \beta * f_2$, where $f_1$is the first leaner, β is some value, and f_2, the proposed second leaner, For all leaner, M.

$$\beta_m = argmin\, \beta\, (\sum_{i=1}^{M} \quad L(y, f_1(x) + \beta f_2(x))$$

When can then say after 1 iteration, that our model is now:

$$F_2(x) = \; f_1(x) + \beta_m f_2$$

We then repeat this process as many times as we can to gain a better and better prediction each time.

Just like with cox regression and Random Survival Forests we are evaluating the strength of our model with the concordance index. This is to have a direct comparison with the two other models, allowing us to easily compare the 3 to find which one is the best at predicting. We are also finding the most important variables to see which specific variables have the most impact on survivability.

It is now time to find the most optimal parameters for Random Survival Forests and Gradient Boosting Machine. Achieving this isn't as simple as it sounds as no model can be tuned in the same way, as depending on which parameters you use, the model can under or overfit the data. If the model is underfitted, the characteristics of the data have not been properly realised and the predictions will be very inaccurate, if overfitting happens, the model has too rigorously fit to the training data, achieving a very high performance on the training data, but will not translate to the different data in the test set. To know if a model is underfitted, you must look at the performance of prediction on both the training data and test data, if they are both low, underfitting is likely and the number of estimators in the model should be increased. If overfitting has occurred, the performance of the training set would be very large, but the performance on the test set will be very low, and we must use less estimators in our model. We also have to look at the efficiency of our models, has using a high number of parameters will increase the performance, but also increase the computational time of our program, we need to find a balance between these two factors to make our model has accurate as possible whilst still being efficient.

To check when overfitting occurs an oob improvement test it performed. This test will keep training the data with more and more estimators and record the average improvement, once this average improvement becomes negative it terminates because it has found the level at which overfitting occurs.

For both models a method was devised to check what the best combination of parameters fit our data the best. The parameter with the most effect on accuracy is the n_estimators parameter used in sksurv. In Random Survival Forest the n_estimators is the number of trees in the forest, and in Gradient Boosting Machine n_estimators is the amount of leaners. Clearly increasing this parameter will make the model more complex allow it to detect finer intricacies in the data and produce a more accurate model, however increasing the complexity will also increase the computational efficiency. To find the best balance, we ran our models' multiple times, increasing the amount of n_estimators each time, and recording their achieved concordance index's. We then plotted these results to see the how accuracy was changing by increasing n_estimators and find an optimum.

As running the machine learning algorithms multiple times takes a very long time, finding the optimum in the other parameters was not realistic. We instead ran each parameter at a low value and a high value and recorded the concordance index. We then plotted a graph of each combination of low parameters and high parameters seeing which one gave us the best concordance index.

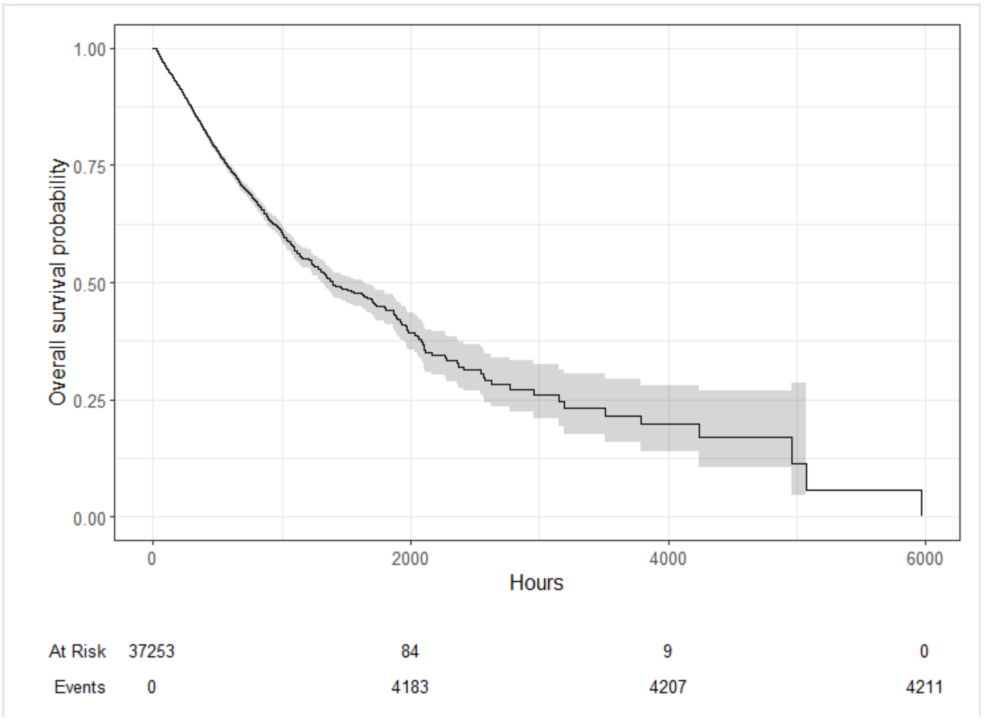
3.6 Pre-processing

Before any algorithm could be implemented pre-processing steps had to be taken. In our case we started by imputing the database this is not necessary for algorithms as they do not have assumptions on the "fullness" of the data, however all algorithms have a statistically significant improvement after imputation (Hong, S., & Lynn, H. S. (2020). To keep the sanctity of the accuracy of the database intact, a missingness threshold was also applied here, dropping the variables with a large amount of missingness in them. The imputation in both cases was done to a mean level, meaning the data points that were missing were replaced by the mean of that specific column. Through sklearn, imputation can be achieved by inputting either, mean, median, most frequent, or constant values into the missing values. There is no evidence to suggest that one method gives an increase in prediction accuracy, so we decided to choose to mean level. An assumption present in both algorithms is that all values in the dataset have to be numeric, this meant that the "ethnicity" variable had to be removed from the dataset as it helps character elements. This variable had very little effect on the survival function, so removing this at this stage will not statistically significantly change the results. The variables removed through this process are, `'ck_amax', 'ck_amin', 'ck_kurtosis', 'ck_mean', 'ck_skew', 'ck_std', 'crp_amax', 'crp_amin', 'crp_kurtosis', 'crp_mean', 'crp_skew', 'crp_std', 'o2flow_amax', 'o2flow_amin', 'o2flow_kurtosis', 'o2flow_mean', 'o2flow_skew', 'o2flow_std'.` Other variables weren't related to the vitals of the patient, so they had to be removed, these were, 'inicu_los_h', 'death_after_icu_h', 'death_after_disch_h'. The final pre-processing step was to scale the data. This is done because in machine learning algorithms, if the values of the variables are closer together the algorithm can make better connections in the data, and make these connections faster, increasing the efficiency and performance of the algorithm. Thus, scaling is implemented to normalise the data, reducing the variance in the data points. This was important in our case as, our dataset holds a wide range of data, for example having mean values, minimums and maximums.

4 RESULTS AND DISCUSSION

4.1 KAPLAN-MEIER CURVES

The first plot created was the Kaplan Meier curve of the whole dataset.



	0	2000	4000	6000
At Risk	37253	84	9	0
Events	0	4183	4207	4211

Figure 5: Kaplan Meier Curve

When plotting Kaplan-Meier curve with the ggsurvfit, summary statistics at the bottom. The “At Risk” row indicates the amount of patients at risk and the “events” row corresponds to the amount of events that have taken place, in our case how many patients have died.

As you can see in our curve the survival rate only reaches 0% at 6000 hours, meaning everyone in the study had either died or left the ICU by 6000 hours. However in the last 1000 hours of the study, the survival probability stays stagnant, at around 0.0564%, this is suggesting that at the end of the study 0 patients died, and by the end all the patients had been omitted from the ICU. This means that if we are calculating survival time, we should omit these patients from our results as they didn’t die and would skew the data in favour of patients surviving longer. When taking a closer look at the dataset we see this to be true.

When looking at a Kaplan-Meier curve created by ggsurvfit, we see a black line, corresponding to the predicted survival probability, surrounded by a grey shaded area. The shaded area in these curves represents the 95% confidence interval on that survival probability prediction, therefore for each corresponding x value, the y value will be inside that shaded region, with a 95% confidence.

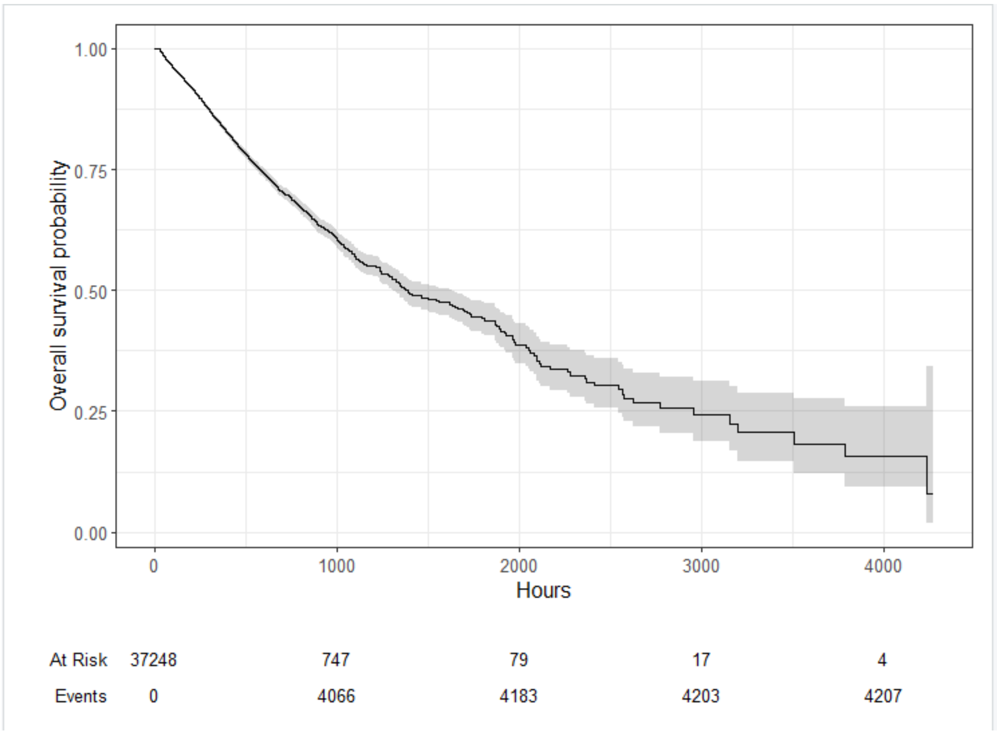


Figure 6: Kaplan Meier Curve

Thus, we plotted another Kaplan-Meier curve, but stopping at 4500. This curve gives a better representation of the data.

To understand this survival function more clearly, ggsurvfit was used to calculate different statistics of the curve. When splitting survival probability into quartiles, we found:

Characteristic	Lower quartile survival (95% CI)
Overall	575 (550, 602)

75% survival probability appeared at 575 hours.

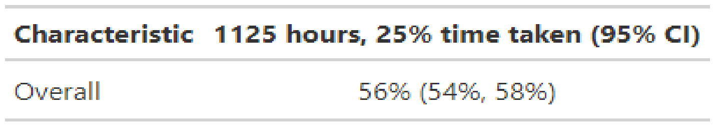
Characteristic	Median survival (95% CI)
Overall	1,390 (1,321, 1,644)

50% survival probability appeared at 1390 hours.

Characteristic	Upper Quartile survival (95% CI)
Overall	3,153 (2,575, —)

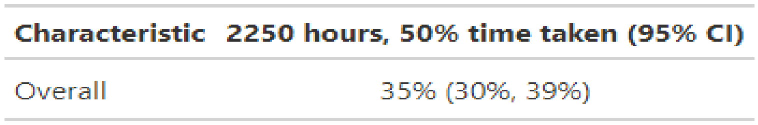
25% survival probability appeared at 3153 hours.

Conversely, when splitting time into quartiles, we found:



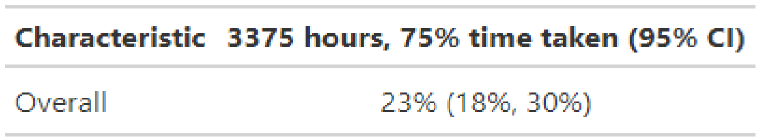
Characteristic	1125 hours, 25% time taken (95% CI)
Overall	56% (54%, 58%)

At 1125 hours, a quarter of the way through the study, the survival probability of a given patient is 56%.



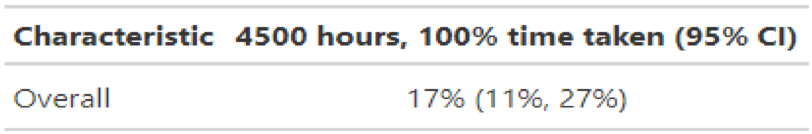
Characteristic	2250 hours, 50% time taken (95% CI)
Overall	35% (30%, 39%)

At 2250 hours, halfway through the study, the survival probability of a given patient is 35%.



Characteristic	3375 hours, 75% time taken (95% CI)
Overall	23% (18%, 30%)

At 3375 hours, 75% of the way through the study the survival probability of a patient is 23%.



Characteristic	4500 hours, 100% time taken (95% CI)
Overall	17% (11%, 27%)

At the end of the study, 17% of the patients were still alive.

This shows that the most dangerous period for a patient is when they are first omitted to the ICU, after this period the death rate significantly slows, 44% of patients died in the first quarter of the study, whereas in the second quarter 21% died, meaning the death rate decreased by 23%. A similar trend follows in the later quarters of the study, as is the third quarter the survival probability reduced by 12% and in the final quarter by 6%. Showing that for every 1125 hours, a quarter of the study, the survival probability reduces by a half.

4.2 COX REGRESSION

Once the characteristics of the dataset were better understood it was time to complete the cox regression. We first completed cox regression model on each category of variable as described in the methods section. The outputs of which are below the table of results.

There are 3 sections of the cox regression output that are important to us in this analysis. The first and most important part of the output is the concordance index. The concordance index is the fraction of comparable cases for which the predicted and observed order of event times agree. In other words, it is the percentage of correct predictions or performance of the model. If the prediction were completely random, you would assume a concordance index of 0.5, thus we are looking for a concordance index of much greater than that. The second aspect of importance is the hazard ratio, given in R as the exp(coef) column. This describes if the given variable as a positive or negative relation to risk of the patient. If the value is greater than 1 then it has a positive effect on risk if it is less than 1 it has a negative effect on risk. The final part of the output of importance are the p values. These show how statistically significant the result is, in this study we will be taking the critical value at a significance level of 0.5. In the R output, the p values are categorically ranked between 5 levels of significance, in our study we will rank any in the top 3 categories as statistically significant.

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

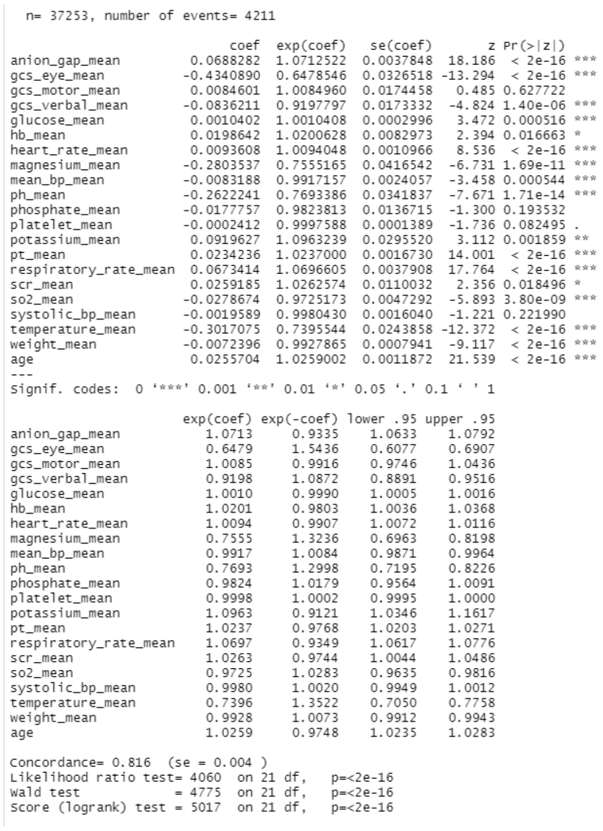
	Concordance	Positive	Negative	Significant	Not Significant
Mean	0.816	Anion_gap Gcs_motor Glucose Hb Heart_rate Potassium Pt Respiratory_rate Scr Age	Gcs_eye Gcs_verbal Magnesium Mean_bp Ph Phosphate Platelet So2 Systolic_bp Temperature Weight	Anion_gap Gcs_eye Gcs_verbal Glucose Hb Heart_rate Magnesium Mean_bp Potassium Pt Respiratory_rate Scr So2 Temperature Weight Age	Gcs_motor Phosphate Platelet Systolic_bp
Max	0.823	Anion_gap Heart_rate Mean_bp Phosphate Pt Respiratory_rate Age	Gcs_eye Gcs_motor Gcs_verbal Glucose Hb Magnesium Ph Platelet Potassium Scr So2 Systolic_bp Temperature Weight	Anion_gap Heart_rate Phosphate Pt Respiratory_rate Age Gcs_eye Gcs_motor Gcs_verbal Glucose Hb Magnesium Ph Platelet Potassium So2 Systolic_bp Temperature Weight	Scr Mean_bp
Min	0.784	Anion_gap Glucose Hb Heart rate Potassium Pt	gcs_motor gcs_verbal magnesium mean_bp ph phosphate	Anion_gap Glucose Hb Heart rate Potassium Pt	gcs_motor phosphate Scr

		Respiratory_rate Scr Age	platelet so2 systolic temperature weight	Respiratory_rate Age gcs_verbal magnesium mean_bp ph platelet so2 systolic temperature weight	
Kurtosis	0.644	Anion_gap gcs_motor heart_rate magnesium mean_bp ph phosphate platelet respiratory_rate scr systlolic_bp temperature age	Gcs_eye Gcs_verbal Glucose Hb Potassium Pt weight	Anion_gap Gcs_motor Gcs_verbal Hb Mean_bp Ph Phosphate Platelet Potassium Pt Respiratory_rate Scr So2 Systolic_bp Temperature Weight Age	Gcs_eye Glucose Heart_rate Magnesium
Skew	0.666	Gcs_eye Gcs_motor Hb Magnesium Mean_bp Phosphate Platelet Scr Systolic_bp age	Anion_gap Gcs_verbal Glucose Heart_rate Ph Potassium Pt Respiratory_rate So2 Temperature weight	Gcs_eye Gcs_motor Gcs_verbal Hb Magnesium Ph Phosphate Pt Respiratory_rate So2 Systolic_bp Weight age	Anion_gap Glucose Heart_rate Mean_bp Platelet Potassium Scr Temperature
Standard Deviation	0.729	Anion_gap Gcs_motor Glucose Heart_rate Mean_bp Ph	Gcs_eye Gcs_verbal Hb Scr weight	Anion_gap Gcs_eye Gcs_verbal Hb Heart_rate Mean_bp	Gcs_motor Glucose Magnesium Potassium Scr

		Phosphate Platelet Pt Respiratory So2 Systolic_bp Temperature age		Ph Phosphate Platelet Pt Respiratory_rate So2 Systolic_bp Temperature Weight age	

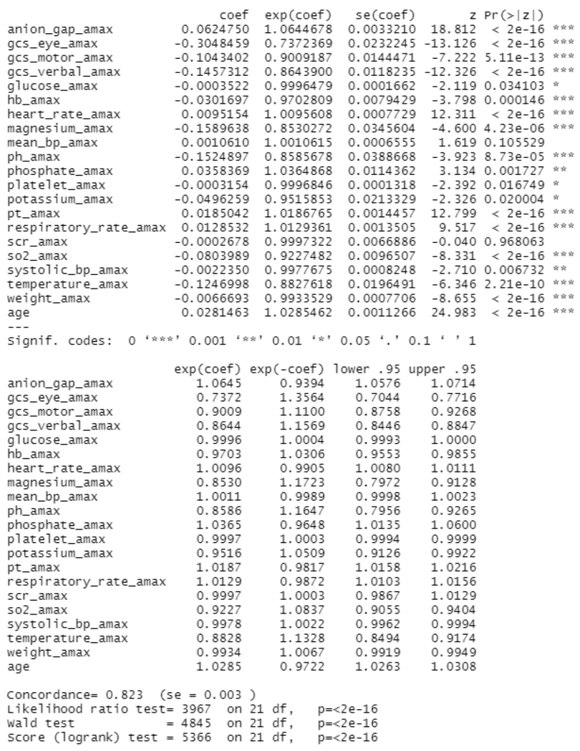
From this we found that the variables that positively affected the performance of the model the most was the maximum variables, and the worst being the skew variables.

Mean



```
  n= 37253, number of events= 4211

                          coef  exp(coef)   se(coef)       z Pr(>|z|)    
anion_gap_mean       0.0688282  1.0712522  0.0037848  18.186  < 2e-16 ***
gcs_eye_mean        -0.4340890  0.6478546  0.0326518 -13.294  < 2e-16 ***
gcs_motor_mean       0.0084601  1.0084960  0.0174458   0.485 0.627722    
gcs_verbal_mean     -0.0836211  0.9197797  0.0173332  -4.824 1.40e-06 ***
glucose_mean         0.0010402  1.0010408  0.0002996   3.472 0.000516 ***
hb_mean              0.0198642  1.0200628  0.0082973   2.394 0.016663 *  
heart_rate_mean      0.0093608  1.0094048  0.0010966   8.536  < 2e-16 ***
magnesium_mean      -0.2803537  0.7555165  0.0416542  -6.731 1.69e-11 ***
mean_bp_mean        -0.0083188  0.9917157  0.0024057  -3.458 0.000544 ***
ph_mean             -0.2622241  0.7693386  0.0341837  -7.671 1.71e-14 ***
phosphate_mean      -0.0177757  0.9823813  0.0136715  -1.300 0.193532    
platelet_mean       -0.0002412  0.9997588  0.0001389  -1.736 0.082495 .  
potassium_mean       0.0919627  1.0963239  0.0295520   3.112 0.001859 ** 
pt_mean              0.0234236  1.0237000  0.0016730  14.001  < 2e-16 ***
respiratory_rate_mean 0.0673414  1.0696605  0.0037908  17.764  < 2e-16 ***
scr_mean             0.0259185  1.0262574  0.0110032   2.356 0.018496 *  
so2_mean            -0.0278674  0.9725173  0.0047292  -5.893 3.80e-09 ***
systolic_bp_mean    -0.0019589  0.9980430  0.0016040  -1.221 0.221990    
temperature_mean    -0.3017075  0.7395544  0.0243858 -12.372  < 2e-16 ***
weight_mean         -0.0072396  0.9927865  0.0007941  -9.117  < 2e-16 ***
age                  0.0255704  1.0259002  0.0011872  21.539  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                      exp(coef) exp(-coef) lower .95 upper .95
anion_gap_mean           1.0713     0.9335    1.0633    1.0792
gcs_eye_mean             0.6479     1.5436    0.6077    0.6907
gcs_motor_mean           1.0085     0.9916    0.9746    1.0436
gcs_verbal_mean          0.9198     1.0872    0.8891    0.9516
glucose_mean             1.0010     0.9990    1.0005    1.0016
hb_mean                  1.0201     0.9803    1.0036    1.0368
heart_rate_mean          1.0094     0.9907    1.0072    1.0116
magnesium_mean           0.7555     1.3236    0.6963    0.8198
mean_bp_mean             0.9917     1.0084    0.9871    0.9964
ph_mean                  0.7693     1.2998    0.7195    0.8226
phosphate_mean           0.9824     1.0179    0.9564    1.0091
platelet_mean            0.9998     1.0002    0.9995    1.0000
potassium_mean           1.0963     0.9121    1.0346    1.1617
pt_mean                  1.0237     0.9768    1.0203    1.0271
respiratory_rate_mean    1.0697     0.9349    1.0617    1.0776
scr_mean                 1.0263     0.9744    1.0044    1.0486
so2_mean                 0.9725     1.0283    0.9635    0.9816
systolic_bp_mean         0.9980     1.0020    0.9949    1.0012
temperature_mean         0.7396     1.3522    0.7050    0.7758
weight_mean              0.9928     1.0073    0.9912    0.9943
age                      1.0259     0.9748    1.0235    1.0283

Concordance= 0.816  (se = 0.004 )
Likelihood ratio test= 4060  on 21 df,   p=<2e-16
Wald test            = 4775  on 21 df,   p=<2e-16
Score (logrank) test = 5017  on 21 df,   p=<2e-16
```

amax



```
                          coef  exp(coef)   se(coef)       z Pr(>|z|)    
anion_gap_amax       0.0624750  1.0644678  0.0033210  18.812  < 2e-16 ***
gcs_eye_amax        -0.3048459  0.7372369  0.0232245 -13.126  < 2e-16 ***
gcs_motor_amax      -0.1043402  0.9009187  0.0144471  -7.222 5.11e-13 ***
gcs_verbal_amax     -0.1457312  0.8643900  0.0118235 -12.326  < 2e-16 ***
glucose_amax        -0.0003522  0.9996479  0.0001662  -2.119 0.034103 *  
hb_amax             -0.0301697  0.9702809  0.0079429  -3.798 0.000146 ***
heart_rate_amax      0.0095154  1.0095608  0.0007729  12.311  < 2e-16 ***
magnesium_amax      -0.1589638  0.8530272  0.0345604  -4.600 4.23e-06 ***
mean_bp_amax         0.0010610  1.0010615  0.0006555   1.619 0.105529    
ph_amax             -0.1524897  0.8585678  0.0388668  -3.923 8.73e-05 ***
phosphate_amax       0.0358369  1.0364868  0.0114362   3.134 0.001727 ** 
platelet_amax       -0.0003154  0.9996846  0.0001318  -2.392 0.016749 *  
potassium_amax      -0.0496259  0.9515853  0.0213329  -2.326 0.020004 *  
pt_amax              0.0185042  1.0186765  0.0014457  12.799  < 2e-16 ***
respiratory_rate_amax 0.0128532  1.0129361  0.0013505   9.517  < 2e-16 ***
scr_amax            -0.0002678  0.9997322  0.0066886  -0.040 0.968063    
so2_amax            -0.0803989  0.9227482  0.0096507  -8.331  < 2e-16 ***
systolic_bp_amax    -0.0022350  0.9977675  0.0008248  -2.710 0.006732 ** 
temperature_amax    -0.1246998  0.8827618  0.0196491  -6.346 2.21e-10 ***
weight_amax         -0.0066693  0.9933529  0.0007706  -8.655  < 2e-16 ***
age                  0.0281463  1.0285462  0.0011266  24.983  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                      exp(coef) exp(-coef) lower .95 upper .95
anion_gap_amax           1.0645     0.9394    1.0576    1.0714
gcs_eye_amax             0.7372     1.3564    0.7044    0.7716
gcs_motor_amax           0.9009     1.1100    0.8758    0.9268
gcs_verbal_amax          0.8644     1.1569    0.8446    0.8847
glucose_amax             0.9996     1.0004    0.9993    1.0000
hb_amax                  0.9703     1.0306    0.9553    0.9855
heart_rate_amax          1.0096     0.9905    1.0080    1.0111
magnesium_amax           0.8530     1.1723    0.7972    0.9128
mean_bp_amax             1.0011     0.9989    0.9998    1.0023
ph_amax                  0.8586     1.1647    0.7956    0.9265
phosphate_amax           1.0365     0.9648    1.0135    1.0600
platelet_amax            0.9997     1.0003    0.9994    0.9999
potassium_amax           0.9516     1.0509    0.9126    0.9922
pt_amax                  1.0187     0.9817    1.0158    1.0216
respiratory_rate_amax    1.0129     0.9872    1.0103    1.0156
scr_amax                 0.9997     1.0003    0.9867    1.0129
so2_amax                 0.9227     1.0837    0.9055    0.9404
systolic_bp_amax         0.9978     1.0022    0.9962    0.9994
temperature_amax         0.8828     1.1328    0.8494    0.9174
weight_amax              0.9934     1.0067    0.9919    0.9949
age                      1.0285     0.9722    1.0263    1.0308

Concordance= 0.823  (se = 0.003 )
Likelihood ratio test= 3967  on 21 df,   p=<2e-16
Wald test            = 4845  on 21 df,   p=<2e-16
Score (logrank) test = 5366  on 21 df,   p=<2e-16
```

amin

```
                          coef  exp(coef)   se(coef)       z Pr(>|z|)    
anion_gap_amin         0.0896468  1.0937879  0.0040621  22.069  < 2e-16 ***
gcs_eye_amin          -0.2490166  0.7795670  0.0272412  -9.141  < 2e-16 ***
gcs_motor_amin        -0.0077539  0.9922761  0.0122227  -0.634 0.525833    
gcs_verbal_amin       -0.0577449  0.9438907  0.0165702  -3.485 0.000492 ***
glucose_amin           0.0041187  1.0041272  0.0003913  10.526  < 2e-16 ***
hb_amin                0.0641087  1.0662083  0.0075088   8.538  < 2e-16 ***
heart_rate_amin        0.0102637  1.0103166  0.0010312   9.953  < 2e-16 ***
magnesium_amin        -0.2385608  0.7877608  0.0401373  -5.944 2.79e-09 ***
mean_bp_amin          -0.0109728  0.9890872  0.0020055  -5.471 4.47e-08 ***
ph_amin               -0.1579369  0.8539037  0.0210095  -7.517 5.59e-14 ***
phosphate_amin        -0.0097108  0.9903362  0.0148591  -0.654 0.513417    
platelet_amin         -0.0005534  0.9994467  0.0001492  -3.709 0.000208 ***
potassium_amin         0.1148339  1.1216872  0.0290854   3.948 7.88e-05 ***
pt_amin                0.0249192  1.0252323  0.0017900  13.921  < 2e-16 ***
respiratory_rate_amin  0.0417472  1.0426309  0.0036020  11.590  < 2e-16 ***
scr_amin               0.0231987  1.0234699  0.0125025   1.856 0.063522 .  
so2_amin              -0.0061957  0.9938234  0.0011290  -5.488 4.07e-08 ***
systolic_bp_amin      -0.0078725  0.9921584  0.0014743  -5.340 9.29e-08 ***
temperature_amin      -0.2056157  0.8141459  0.0172197 -11.941  < 2e-16 ***
weight_amin           -0.0065978  0.9934239  0.0007736  -8.529  < 2e-16 ***
age                    0.0234400  1.0237169  0.0011830  19.815  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                      exp(coef) exp(-coef) lower .95 upper .95
anion_gap_amin           1.0938     0.9143    1.0851    1.1025
gcs_eye_amin             0.7796     1.2828    0.7390    0.8223
gcs_motor_amin           0.9923     1.0078    0.9688    1.0163
gcs_verbal_amin          0.9439     1.0594    0.9137    0.9750
glucose_amin             1.0041     0.9959    1.0034    1.0049
hb_amin                  1.0662     0.9379    1.0506    1.0820
heart_rate_amin          1.0103     0.9898    1.0083    1.0124
magnesium_amin           0.7878     1.2694    0.7282    0.8522
mean_bp_amin             0.9891     1.0110    0.9852    0.9930
ph_amin                  0.8539     1.1711    0.8195    0.8898
phosphate_amin           0.9903     1.0098    0.9619    1.0196
platelet_amin            0.9994     1.0006    0.9992    0.9997
potassium_amin           1.1217     0.8915    1.0595    1.1875
pt_amin                  1.0252     0.9754    1.0216    1.0288
respiratory_rate_amin    1.0426     0.9591    1.0353    1.0500
scr_amin                 1.0235     0.9771    0.9987    1.0489
so2_amin                 0.9938     1.0062    0.9916    0.9960
systolic_bp_amin         0.9922     1.0079    0.9893    0.9950
temperature_amin         0.8141     1.2283    0.7871    0.8421
weight_amin              0.9934     1.0066    0.9919    0.9949
age                      1.0237     0.9768    1.0213    1.0261

Concordance= 0.784  (se = 0.004 )
Likelihood ratio test= 3363  on 21 df,   p=<2e-16
Wald test            = 3825  on 21 df,   p=<2e-16
Score (logrank) test = 3859  on 21 df,   p=<2e-16
```

Kurtosis

```
                              coef exp(coef)  se(coef)      z Pr(>|z|)    
anion_gap_kurtosis         0.017113  1.017260  0.006732  2.542 0.011016 *  
gcs_eye_kurtosis          -0.007381  0.992646  0.005148 -1.434 0.151679    
gcs_motor_kurtosis         0.025838  1.026175  0.004454  5.801 6.57e-09 ***
gcs_verbal_kurtosis       -0.034665  0.965928  0.005791 -5.986 2.15e-09 ***
glucose_kurtosis          -0.009124  0.990917  0.006579 -1.387 0.165472    
hb_kurtosis               -0.027993  0.972395  0.006902 -4.056 4.99e-05 ***
heart_rate_kurtosis        0.012091  1.012165  0.006877  1.758 0.078732 .  
magnesium_kurtosis         0.008502  1.008538  0.006054  1.404 0.160195    
mean_bp_kurtosis           0.012068  1.012141  0.005700  2.117 0.034240 *  
ph_kurtosis                0.021440  1.021672  0.003502  6.121 9.27e-10 ***
phosphate_kurtosis         0.015192  1.015308  0.006750  2.251 0.024408 *  
platelet_kurtosis          0.014841  1.014951  0.006878  2.158 0.030962 *  
potassium_kurtosis        -0.027363  0.973008  0.006326 -4.326 1.52e-05 ***
pt_kurtosis               -0.020642  0.979569  0.005533 -3.731 0.000191 ***
respiratory_rate_kurtosis  0.023171  1.023441  0.005789  4.003 6.26e-05 ***
scr_kurtosis               0.014968  1.015081  0.006053  2.473 0.013397 *  
so2_kurtosis               0.017426  1.017578  0.002968  5.871 4.33e-09 ***
systolic_bp_kurtosis       0.041760  1.042644  0.009626  4.338 1.44e-05 ***
temperature_kurtosis       0.018852  1.019031  0.008871  2.125 0.033584 *  
weight_kurtosis           -0.006031  0.993988  0.002521 -2.393 0.016732 *  
age                        0.026044  1.026386  0.001109 23.477  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                          exp(coef) exp(-coef) lower .95 upper .95
anion_gap_kurtosis           1.0173     0.9830    1.0039    1.0308
gcs_eye_kurtosis             0.9926     1.0074    0.9827    1.0027
gcs_motor_kurtosis           1.0262     0.9745    1.0173    1.0352
gcs_verbal_kurtosis          0.9659     1.0353    0.9550    0.9770
glucose_kurtosis             0.9909     1.0092    0.9782    1.0038
hb_kurtosis                  0.9724     1.0284    0.9593    0.9856
heart_rate_kurtosis          1.0122     0.9880    0.9986    1.0259
magnesium_kurtosis           1.0085     0.9915    0.9966    1.0206
mean_bp_kurtosis             1.0121     0.9880    1.0009    1.0235
ph_kurtosis                  1.0217     0.9788    1.0147    1.0287
phosphate_kurtosis           1.0153     0.9849    1.0020    1.0288
platelet_kurtosis            1.0150     0.9853    1.0014    1.0287
potassium_kurtosis           0.9730     1.0277    0.9610    0.9851
pt_kurtosis                  0.9796     1.0209    0.9690    0.9902
respiratory_rate_kurtosis    1.0234     0.9771    1.0119    1.0351
scr_kurtosis                 1.0151     0.9851    1.0031    1.0272
so2_kurtosis                 1.0176     0.9827    1.0117    1.0235
systolic_bp_kurtosis         1.0426     0.9591    1.0232    1.0625
temperature_kurtosis         1.0190     0.9813    1.0015    1.0369
weight_kurtosis              0.9940     1.0060    0.9891    0.9989
age                          1.0264     0.9743    1.0242    1.0286

Concordance= 0.644  (se = 0.005 )
Likelihood ratio test= 908.9  on 21 df,   p=<2e-16
Wald test            = 874.5  on 21 df,   p=<2e-16
Score (logrank) test = 882.7  on 21 df,   p=<2e-16
```

Skew

```
                           coef  exp(coef)   se(coef)       z Pr(>|z|)    
anion_gap_skew        -0.0038180  0.9961893  0.0160774  -0.237 0.812288    
gcs_eye_skew           0.1041900  1.1098113  0.0150999   6.900 5.20e-12 ***
gcs_motor_skew         0.0440099  1.0449927  0.0161858   2.719 0.006547 ** 
gcs_verbal_skew       -0.0631038  0.9388460  0.0160330  -3.936 8.29e-05 ***
glucose_skew          -0.0238162  0.9764652  0.0163865  -1.453 0.146112    
hb_skew                0.0372172  1.0379184  0.0153004   2.432 0.014998 *  
heart_rate_skew       -0.0034268  0.9965790  0.0184672  -0.186 0.852787    
magnesium_skew         0.0420341  1.0429300  0.0153745   2.734 0.006257 ** 
mean_bp_skew           0.0003909  1.0003910  0.0184617   0.021 0.983108    
ph_skew               -0.1300684  0.8780353  0.0121614 -10.695  < 2e-16 ***
phosphate_skew         0.0519023  1.0532729  0.0158477   3.275 0.001056 ** 
platelet_skew          0.0305047  1.0309748  0.0155855   1.957 0.050319 .  
potassium_skew        -0.0116427  0.9884248  0.0146592  -0.794 0.427064    
pt_skew               -0.0855743  0.9179850  0.0154487  -5.539 3.04e-08 ***
respiratory_rate_skew -0.1666580  0.8464891  0.0153122 -10.884  < 2e-16 ***
scr_skew               0.0212206  1.0214473  0.0167122   1.270 0.204169    
so2_skew              -0.0875078  0.9162117  0.0114874  -7.618 2.58e-14 ***
systolic_bp_skew       0.0951010  1.0997700  0.0245276   3.877 0.000106 ***
temperature_skew      -0.0235952  0.9766810  0.0186555  -1.265 0.205950    
weight_skew           -0.0301136  0.9703352  0.0093845  -3.209 0.001333 ** 
age                    0.0264144  1.0267663  0.0011165  23.658  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                      exp(coef) exp(-coef) lower .95 upper .95
anion_gap_skew           0.9962     1.0038    0.9653    1.0281
gcs_eye_skew             1.1098     0.9011    1.0774    1.1431
gcs_motor_skew           1.0450     0.9569    1.0124    1.0787
gcs_verbal_skew          0.9388     1.0651    0.9098    0.9688
glucose_skew             0.9765     1.0241    0.9456    1.0083
hb_skew                  1.0379     0.9635    1.0073    1.0695
heart_rate_skew          0.9966     1.0034    0.9612    1.0333
magnesium_skew           1.0429     0.9588    1.0120    1.0748
mean_bp_skew             1.0004     0.9996    0.9648    1.0373
ph_skew                  0.8780     1.1389    0.8574    0.8992
phosphate_skew           1.0533     0.9494    1.0211    1.0865
platelet_skew            1.0310     0.9700    1.0000    1.0630
potassium_skew           0.9884     1.0117    0.9604    1.0172
pt_skew                  0.9180     1.0893    0.8906    0.9462
respiratory_rate_skew    0.8465     1.1814    0.8215    0.8723
scr_skew                 1.0214     0.9790    0.9885    1.0555
so2_skew                 0.9162     1.0915    0.8958    0.9371
systolic_bp_skew         1.0998     0.9093    1.0482    1.1539
temperature_skew         0.9767     1.0239    0.9416    1.0131
weight_skew              0.9703     1.0306    0.9527    0.9883
age                      1.0268     0.9739    1.0245    1.0290

Concordance= 0.666  (se = 0.005 )
Likelihood ratio test= 1141  on 21 df,   p=<2e-16
Wald test            = 1103  on 21 df,   p=<2e-16
Score (logrank) test = 1107  on 21 df,   p=<2e-16
```

Standard Deviation

```
                          coef  exp(coef)   se(coef)       z Pr(>|z|)    
anion_gap_std         0.0952329  1.0999150  0.0136020   7.001 2.53e-12 ***
gcs_eye_std          -0.1109576  0.8949767  0.0519167  -2.137   0.0326 *  
gcs_motor_std         0.0004323  1.0004324  0.0285236   0.015   0.9879    
gcs_verbal_std       -0.3525886  0.7028663  0.0299262 -11.782  < 2e-16 ***
glucose_std           0.0007593  1.0007595  0.0005165   1.470   0.1415    
hb_std               -0.2879134  0.7498266  0.0428811  -6.714 1.89e-11 ***
heart_rate_std        0.0130150  1.0131001  0.0032621   3.990 6.61e-05 ***
magnesium_std         0.0547520  1.0562786  0.0895479   0.611   0.5409    
mean_bp_std           0.0126554  1.0127358  0.0031253   4.049 5.14e-05 ***
ph_std                0.4526921  1.5725399  0.0578076   7.831 4.84e-15 ***
phosphate_std         0.4325557  1.5411914  0.0396304  10.915  < 2e-16 ***
platelet_std          0.0027238  1.0027276  0.0011308   2.409   0.0160 *  
potassium_std         0.0016974  1.0016989  0.0699752   0.024   0.9806    
pt_std                0.0718259  1.0744683  0.0056561  12.699  < 2e-16 ***
respiratory_rate_std  0.0206291  1.0208433  0.0088162   2.340   0.0193 *  
scr_std              -0.0283134  0.9720836  0.0430803  -0.657   0.5110    
so2_std               0.0414418  1.0423125  0.0039810  10.410  < 2e-16 ***
systolic_bp_std       0.0081505  1.0081838  0.0036451   2.236   0.0253 *  
temperature_std       0.3931727  1.4816742  0.0501548   7.839 4.53e-15 ***
weight_std           -0.0522310  0.9491096  0.0094875  -5.505 3.69e-08 ***
age                   0.0266748  1.0270337  0.0011150  23.923  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                     exp(coef) exp(-coef) lower .95 upper .95
anion_gap_std           1.0999     0.9092    1.0710    1.1296
gcs_eye_std             0.8950     1.1173    0.8084    0.9908
gcs_motor_std           1.0004     0.9996    0.9460    1.0580
gcs_verbal_std          0.7029     1.4227    0.6628    0.7453
glucose_std             1.0008     0.9992    0.9997    1.0018
hb_std                  0.7498     1.3336    0.6894    0.8156
heart_rate_std          1.0131     0.9871    1.0066    1.0196
magnesium_std           1.0563     0.9467    0.8862    1.2589
mean_bp_std             1.0127     0.9874    1.0066    1.0190
ph_std                  1.5725     0.6359    1.4041    1.7612
phosphate_std           1.5412     0.6488    1.4260    1.6657
platelet_std            1.0027     0.9973    1.0005    1.0050
potassium_std           1.0017     0.9983    0.8733    1.1489
pt_std                  1.0745     0.9307    1.0626    1.0864
respiratory_rate_std    1.0208     0.9796    1.0034    1.0386
scr_std                 0.9721     1.0287    0.8934    1.0577
so2_std                 1.0423     0.9594    1.0342    1.0505
systolic_bp_std         1.0082     0.9919    1.0010    1.0154
temperature_std         1.4817     0.6749    1.3430    1.6347
weight_std              0.9491     1.0536    0.9316    0.9669
age                     1.0270     0.9737    1.0248    1.0293

Concordance= 0.729  (se = 0.004 )
Likelihood ratio test= 1939  on 21 df,   p=<2e-16
Wald test            = 2102  on 21 df,   p=<2e-16
Score (logrank) test = 2157  on 21 df,   p=<2e-16
```

Finally, we completed a larger cox regression model of the dataset as a whole. From this we had an overall concordance index of 0.843.

Positive effect on risk	Negative effect on risk	Significant	Not significant
Magnesium mean	Anion gap mean	Gcs eye mean	Anion gap mean
Platelet mean	Gcs eye mean	Mean bp mean	Gcs motor mean
Potassium mean	Gcs motor mean	Phosphate mean	Gcs verbal mean
Pt mean	Gcs verbal mean	Platelet mean	Glucose mean
Respiratory rate mean	Glucose mean	Respiratory rate mean	Hb mean
Systolic bp mean	Hb mean	So2 mean	Heart rate mean
Weight mean	Heart rate mean	Temperature mean	Magnesium mean
Anion gap max	Mean bp mean	Weight mean	Ph mean
Gcs motor max	Ph mean	Anion gap max	Potassium mean
Heartrate max	Phosphate mean	Gcs eye max	Pt mean
Pt max	Scr mean	Gcs verbal max	Scr mena
Scr max	So2 mean	Glucose max	Systolic bp mean
So2 max	Temperature mean	Hb max	Gcs motor max
Tempurature max	Gcs eye max	Heart rate max	Magnesium max
Anion gap min	Gcs verbal max	Platelet max	Mean bp max
Gcs eye min	Glucose max	Potassium max	Ph max
Gcs verbal min	Hb max	Respiratory rate max	Phosphate max
Glucose min	Magnesium max	Scr max	Pt max
Hb min	Mean bp max	Weight max	So2 max
Heart rate min	Ph max	Gcs motor min	Systolic bp max
Mean bp min	Phosphate max	Gcs verbal min	Temperature max
Ph min	Platelet max	Glucose min	Anion gap min
Phosphate min	Potassium max	Hb min	Gcs eye min
Potassium min	Respiratory rate max	Phosphate min	Heart rate min
Respiratory rate min	Systolic bp max	Platelet min	Magnesium min
Weight min	Weight max	Potassium min	Mean bp min
Anion gap kurtosis	Gcs motor min	So2 min	Pt min
Gcs eye kurtosis	Magnesium min	Gcs verbal kurtosis	Respiratory rate min
Gcs motor kurtosis	Platelet min	Glucose kurtosis	Scr min
Gcs verbal kurtosis	Pt min	Pt kurtosis	Systolic bp min
Glucose kurtosis	Scr min	So2 kurtosis	Temperature min
Magnesium kurtosis	So2 min	Systolic bp kurtosis	Weight min
Ph kurtosis	Systolic bp min	Weight kurtosis	Anion gap kurtosis
Phosphate kurtosis	Temperature min	Heart rate skew	Gcs eye kurtosis
Platelet kurtosis	Hb kurtosis	Phosphate skew	Hb kurtosis
Potassium kurtosis	Heart rate kurtosis	Platelet skew	Heart rate kurtosis
Respiratory rate	Mean bp kurtosis	Respiratory rate skew	Magnesium kurtosis
kurtosis	Pt kurtosis	So2 skew	Mean bp kurtosis
Scr kurtosis	So2 kurtosis	Anion gap std	Ph kurtosis
Systolic bp kurtosis	Temperature kurtosis	Gcs motor std	Phosphate kurtosis
Gcs motor skew	Weight kurtosis	Gcs verbal std	Platelet kurtosis
Glucose skew	Anion gap skew	Glucose std	Potassium kurtosis
Hb skew	Gcs eye skew	Hb std	Respiratory rate
Magnesium skew	Gcs verbal skew	Heart rate std	kurtosis
Platelet skew	Heart rate skew	Mean bp std	Scr kurtosis
Potassium skew	Mean bp skew	Ph std	Temperature kurtosis
Respiratory rate skew	Ph skew	Phosphate std	Anion gap skew
Systolic bp skew	Phosphate skew	Potassium std	Gcs eye skew

Weight skew Gcs eye std Gcs verbal std Glucose std Hb std Magnesium std Mean bp std Ph std Phosphate std Platelet std Potassium std Pt std Respiratory rate std Systolic bp std Weight std age	Pt skew Scr skew So2 skew Temperature skew Anion gap std Gcs motor std Heart rate std Scr std So2 std Temperature std	Scr std So2 std Weight std age	Gcs motor skew Gcs verbal skew Glucose skew Hb skew Magnesium skew Mean bp skew Ph skew Potassium skew Pt skew Scr skew Systolic bp skew Temperature skew Weight skew Gcs eye std Magnesium std Platelet std Pt std Respiratory rate std Systolic bp std Temperature std

```
                                coef  exp(coef)   se(coef)      z Pr(>|z|)
anion_gap_mean            -0.0499709  0.9512571  0.0269917 -1.851 0.064120 .
gcs_eye_mean              -0.2136761  0.8076100  0.1002956 -2.130 0.033133 *
gcs_motor_mean            -0.0085638  0.9914728  0.0557555 -0.154 0.877928
gcs_verbal_mean           -0.1076193  0.8979694  0.0760373 -1.415 0.156967
glucose_mean              -0.0007073  0.9992929  0.0011339 -0.624 0.532756
hb_mean                   -0.0368014  0.9638675  0.0814152 -0.452 0.651253
heart_rate_mean           -0.0086705  0.9913670  0.0049482 -1.752 0.079731 .
magnesium_mean             0.0478475  1.0490106  0.2223646  0.215 0.829630
mean_bp_mean              -0.0110443  0.9890164  0.0048968 -2.255 0.024107 *
ph_mean                   -0.2415397  0.7854176  0.1248150 -1.935 0.052968 .
phosphate_mean            -0.4404850  0.6437241  0.0918031 -4.798 1.60e-06 ***
platelet_mean              0.0083471  1.0083821  0.0022600  3.693 0.000221 ***
potassium_mean             0.2278355  1.2558787  0.1401855  1.625 0.104111
pt_mean                    0.0218182  1.0220579  0.0166574  1.310 0.190257
respiratory_rate_mean      0.0672314  1.0695430  0.0104594  6.428 1.29e-10 ***
scr_mean                  -0.1942373  0.8234624  0.1337335 -1.452 0.146384
so2_mean                  -0.0558090  0.9457198  0.0112211 -4.974 6.57e-07 ***
systolic_bp_mean           0.0090308  1.0090717  0.0057683  1.566 0.117443
temperature_mean          -0.2262833  0.7974921  0.0992651 -2.280 0.022632 *
weight_mean                0.0617892  1.0637380  0.0128993  4.790 1.67e-06 ***
anion_gap_amax             0.1007618  1.1060131  0.0272350  3.700 0.000216 ***
gcs_eye_amax              -0.2338820  0.7914552  0.0847329 -2.760 0.005776 **
gcs_motor_amax             0.0278309  1.0282218  0.0609637  0.457 0.648020
gcs_verbal_amax           -0.2088680  0.8115024  0.0859005 -2.432 0.015036 *
glucose_amax              -0.0025881  0.9974152  0.0008306 -3.116 0.001833 **
hb_amax                   -0.2429131  0.7843397  0.0723535 -3.357 0.000787 ***
heart_rate_amax            0.0171328  1.0172804  0.0037147  4.612 3.99e-06 ***
magnesium_amax            -0.1098191  0.8959962  0.1877811 -0.585 0.558665
mean_bp_amax              -0.0035404  0.9964658  0.0034990 -1.012 0.311605
ph_amax                   -0.2571116  0.7732819  0.1817520 -1.415 0.157177
phosphate_amax            -0.0192176  0.9809659  0.0829133 -0.232 0.816709
platelet_amax             -0.0044698  0.9955402  0.0022420 -1.994 0.046192 *
potassium_amax            -0.6289416  0.5331558  0.0980440 -6.415 1.41e-10 ***
pt_amax                    0.0029446  1.0029489  0.0164920  0.179 0.858294
respiratory_rate_amax     -0.0211355  0.9790863  0.0077634 -2.722 0.006480 **
scr_amax                   0.4586295  1.5819044  0.1965672  2.333 0.019638 *
so2_amax                   0.0087934  1.0088322  0.0186437  0.472 0.637174
systolic_bp_amax          -0.0064388  0.9935819  0.0038616 -1.667 0.095442 .
temperature_amax           0.0058742  1.0058915  0.0885428  0.066 0.947105
weight_amax               -0.0857669  0.9178082  0.0170883 -5.019 5.19e-07 ***

anion_gap_amin             0.0221797  1.0224275  0.0274892  0.807 0.419752
gcs_eye_amin               0.0041164  1.0041249  0.0883395  0.047 0.962834
gcs_motor_amin            -0.0989411  0.9057961  0.0483072 -2.048 0.040544 *
gcs_verbal_amin            0.1704363  1.1858221  0.0836080  2.039 0.041498 *
glucose_amin               0.0049909  1.0050034  0.0011575  4.312 1.62e-05 ***
hb_amin                    0.2866582  1.3319688  0.0609194  4.706 2.53e-06 ***
heart_rate_amin            0.0031254  1.0031303  0.0047399  0.659 0.509652
magnesium_amin            -0.0443523  0.9566169  0.2769167 -0.160 0.872751
mean_bp_amin               0.0005086  1.0005088  0.0045291  0.112 0.910581
ph_amin                    0.3595585  1.4326968  0.1190727  3.020 0.002531 **
phosphate_amin             0.4508955  1.5697172  0.1046175  4.310 1.63e-05 ***
platelet_amin             -0.0046870  0.9953239  0.0021017 -2.230 0.025740 *
potassium_amin             0.5483620  1.7304162  0.1404547  3.904 9.45e-05 ***
pt_amin                   -0.0061359  0.9938829  0.0182509 -0.336 0.736724
respiratory_rate_amin      0.0052460  1.0052598  0.0093412  0.562 0.574389
scr_amin                  -0.2528092  0.7766161  0.2064648 -1.224 0.220776
so2_amin                  -0.0162096  0.9839211  0.0050058 -3.238 0.001203 **
systolic_bp_amin          -0.0062246  0.9937947  0.0043281 -1.438 0.150382
temperature_amin          -0.0979252  0.9067167  0.0813395 -1.204 0.228626
weight_amin                0.0181730  1.0183391  0.0125862  1.444 0.148773
anion_gap_kurtosis         0.0026479  1.0026514  0.0080122  0.330 0.741033
gcs_eye_kurtosis           0.0020420  1.0020441  0.0058058  0.352 0.725043
gcs_motor_kurtosis         0.0070358  1.0070606  0.0060188  1.169 0.242413
gcs_verbal_kurtosis        0.0175866  1.0177421  0.0077411  2.272 0.023096 *
glucose_kurtosis           0.0216341  1.0218698  0.0069928  3.094 0.001976 **
hb_kurtosis               -0.0053710  0.9946434  0.0077410 -0.694 0.487788
heart_rate_kurtosis       -0.0025959  0.9974075  0.0096624 -0.269 0.788196
magnesium_kurtosis         0.0023365  1.0023392  0.0069922  0.334 0.738263
mean_bp_kurtosis          -0.0005745  0.9994257  0.0127063 -0.045 0.963940
ph_kurtosis                0.0118677  1.0119384  0.0062873  1.888 0.059084 .
phosphate_kurtosis         0.0069675  1.0069918  0.0078910  0.883 0.377254
platelet_kurtosis          0.0117370  1.0118061  0.0079886  1.469 0.141775
potassium_kurtosis         0.0044624  1.0044724  0.0072512  0.615 0.538288
pt_kurtosis               -0.0120913  0.9879815  0.0060752 -1.990 0.046561 *
respiratory_rate_kurtosis  0.0092406  1.0092834  0.0076755  1.204 0.228625
scr_kurtosis               0.0011668  1.0011675  0.0070248  0.166 0.868076
so2_kurtosis              -0.0356322  0.9649952  0.0087579 -4.069 4.73e-05 ***
systolic_bp_kurtosis       0.0316181  1.0321232  0.0151777  2.083 0.037234 *
temperature_kurtosis      -0.0093955  0.9906485  0.0104054 -0.903 0.366558
weight_kurtosis           -0.0072368  0.9927893  0.0031341 -2.309 0.020941 *
```

```
anion_gap_skew           -0.0129875  0.9870964  0.0216703 -0.599 0.548956
gcs_eye_skew             -0.0193150  0.9808703  0.0269245 -0.717 0.473142
gcs_motor_skew            0.0004235  1.0004236  0.0245702  0.017 0.986248
gcs_verbal_skew          -0.0444135  0.9565583  0.0312758 -1.420 0.155591
glucose_skew              0.0029566  1.0029609  0.0197331  0.150 0.880901
hb_skew                   0.0199372  1.0201372  0.0218113  0.914 0.360678
heart_rate_skew          -0.0642688  0.9377529  0.0325436 -1.975 0.048285 *
magnesium_skew            0.0233939  1.0236697  0.0193412  1.210 0.226455
mean_bp_skew             -0.0328448  0.9676887  0.0366743 -0.896 0.370476
ph_skew                  -0.0376815  0.9630196  0.0204030 -1.847 0.064767 .
phosphate_skew           -0.0426527  0.9582442  0.0198036 -2.154 0.031257 *
platelet_skew             0.0419930  1.0428872  0.0196858  2.133 0.032912 *
potassium_skew            0.0314519  1.0319517  0.0206835  1.521 0.128355
pt_skew                  -0.0059584  0.9940593  0.0192644 -0.309 0.757095
respiratory_rate_skew     0.0626318  1.0646348  0.0256652  2.440 0.014673 *
scr_skew                 -0.0112657  0.9887975  0.0176333 -0.639 0.522894
so2_skew                 -0.1123104  0.8937668  0.0350994 -3.200 0.001375 **
systolic_bp_skew          0.0855925  1.0893623  0.0527061  1.624 0.104385
temperature_skew         -0.0302406  0.9702120  0.0279762 -1.081 0.279722
weight_skew               0.0180143  1.0181776  0.0122338  1.473 0.140884
anion_gap_std            -0.1185871  0.8881744  0.0590717 -2.008 0.044695 *
gcs_eye_std               0.1511343  1.1631529  0.1680779  0.899 0.368550
gcs_motor_std            -0.3933827  0.6747705  0.1106452 -3.555 0.000377 ***
gcs_verbal_std            0.3365569  1.4001185  0.1656258  2.032 0.042150 *
glucose_std               0.0072644  1.0072908  0.0022083  3.290 0.001004 **
hb_std                    0.4431093  1.5575426  0.1399448  3.166 0.001544 **
heart_rate_std           -0.0353345  0.9652824  0.0118041 -2.993 0.002759 **
magnesium_std             0.1621683  1.1760582  0.4834646  0.335 0.737301
mean_bp_std               0.0283079  1.0287124  0.0132124  2.143 0.032152 *
ph_std                    0.8268140  2.2860239  0.3071071  2.692 0.007097 **
phosphate_std             0.6613295  1.9373663  0.1983500  3.334 0.000856 ***
platelet_std              0.0007094  1.0007097  0.0046311  0.153 0.878252
potassium_std             1.4737530  4.3655885  0.2501037  5.893 3.80e-09 ***
pt_std                    0.0038521  1.0038596  0.0356887  0.108 0.914046
respiratory_rate_std      0.0381294  1.0388656  0.0279738  1.363 0.172871
scr_std                  -1.2479351  0.2870970  0.4983782 -2.504 0.012280 *
so2_std                  -0.0716504  0.9308563  0.0200332 -3.577 0.000348 ***
systolic_bp_std           0.0055644  1.0055799  0.0114272  0.487 0.626301
temperature_std          -0.1454590  0.8646254  0.2104963 -0.691 0.489548
weight_std                0.0799956  1.0832823  0.0311017  2.572 0.010109 *
age                       0.0245970  1.0249020  0.0012246 20.085  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Concordance= 0.845  (se = 0.003 )
Likelihood ratio test= 5188  on 121 df,   p=<2e-16
Wald test            = 6033  on 121 df,   p=<2e-16
Score (logrank) test = 6774  on 121 df,   p=<2e-16
```

4.3 Gradient Boosting Machine Parameters

As described in the methods section, when implementing the machine learning algorithms, we had to first find the optimal parameters to have the best performance possible. The first algorithm we ran was a gradient boosting machine. The first parameter we tuned was the n_estimators. We did this by running the algorithm multiple times with an incremented amount of n_estimators each time. We then plotted the concordance indexes to see how it improves by increasing the amount of estimators. Giving this output:

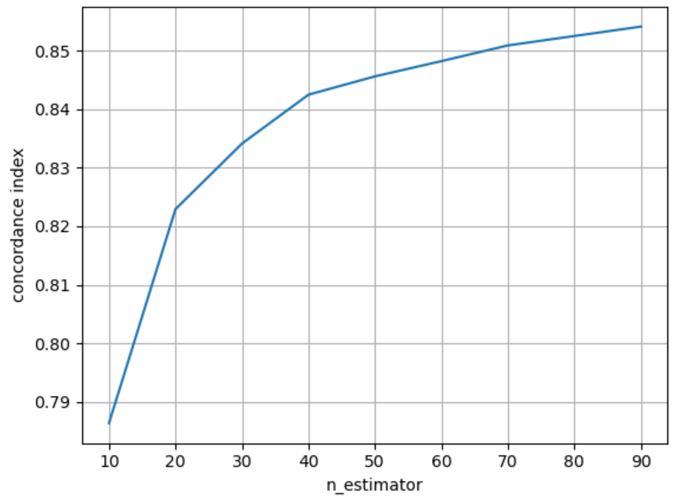


Figure 7: Gradient Boosting Machine n_estimators

The results show that increasing the amount of estimators steadily increases the concordance index of the model, implying that we should use higher number of estimators in our model to achieve the greatest performance. Picking a high number of estimators does however, increase the run time of the algorithm, but in this case, Gradient Boosting machines is an efficient model with a short run time, meaning the time increase is less dramatic. As the highest amount of estimators gave the highest recorded concordance index, it is also suggesting that overfitting is not occurring in the model. Therefore, we have decided to run the machine learning algorithm at 90 estimators.

To prove overfitting did not occure an oob improvement test was completed showing the improvement in performance, by each estimator increase. It showed that after 1000 estimators, overfitting did not occur, thus a high number of estimators should be used. Shown here:

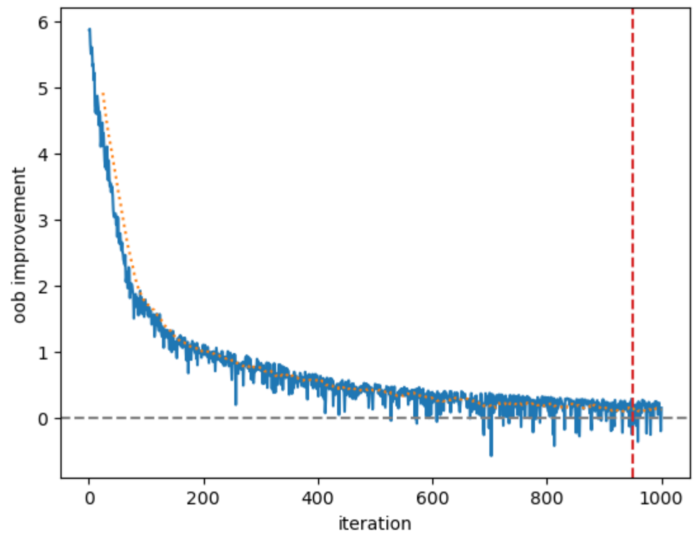
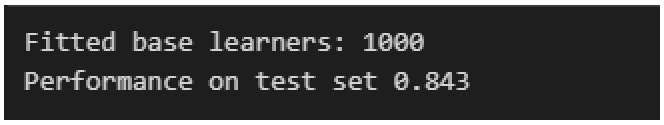


Figure 8: Gradient Boosting Machine OOB Improvement



```
Fitted base learners: 1000
Performance on test set 0.843
```

When completing the same process on component-wise gradient boosting machine, we gained this output:

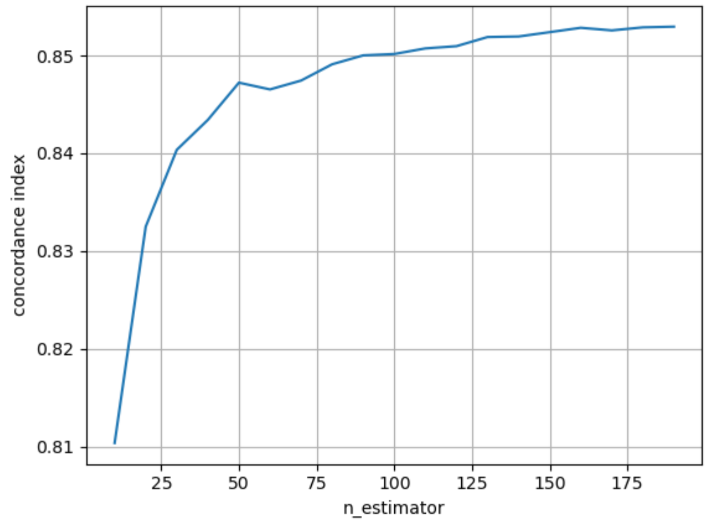
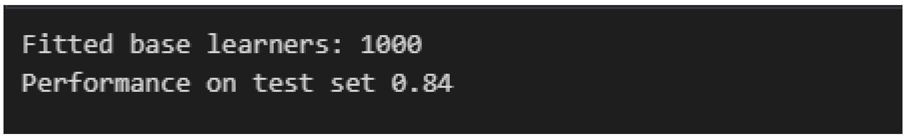


Figure 9: Component-wise n_estimators

As the component wise machine took less time to run each time, it allowed us to run the algorithm up to a higher number of estimators, gaining a more well-rounded understanding of how a high number of estimators affects the performance. Like the normal gradient boosting machine, this graph shows that increasing the number of estimators doesn't affect the performance of the model substantially at all, showing diminishing returns on improvement after 130, thus this is the logical number of estimators to use, as this model takes less time to run, the computational efficiency of the model is kept intact. Again, as the highest number of estimators gave the highest performance overfitting is not occurring.

The oob test when conducted on component-wise analysis gave this output:



```
Fitted base learners: 1000
Performance on test set 0.84
```

This showed that just like in normal gradient boosting machine, overfitting would not occur, thus the improvement graph would look the same as the prior output.

To tune the other parameters in the normal gradient boosted model, we ran the algorithm multiple times again, but this time, differing the other parameters as well. Each different model had a max depth of 1 and a random state of 0, but the ‘no regularisation’ model had a learning rate of 1, the ‘learning rate model had a learning rate of 0.1, the model labelled ‘dropout’ had a learning rate of 1 but a dropout rate of 0.1 and the model labelled subsample, had a learning rate of 1 but a subsample rate of 0.5. When plotting the concordance ratings of these models we had this output:

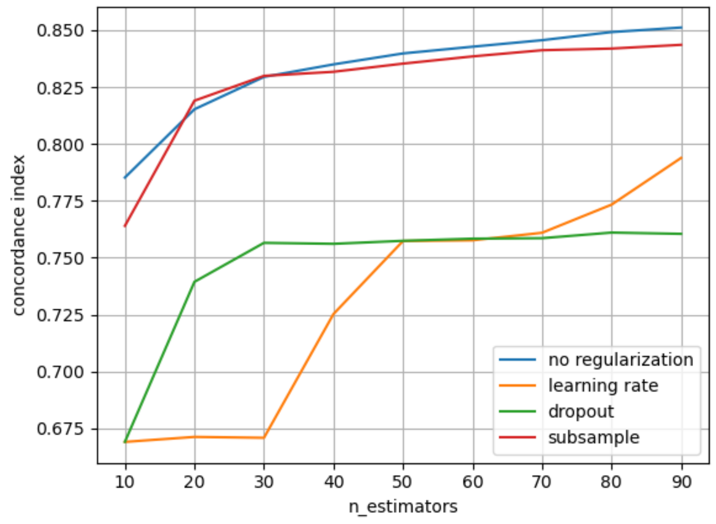


Figure 10: Gradient Boosting Machine Parameters

This shows that the model labelled ‘no regularisation’ is the best model at our chosen level of estimators, (60), therefore we should use a learning rate of 1, and not change the dropout_rate and subsample parameters.

We then did the same process for component-wise gradient boosting machine giving this output:

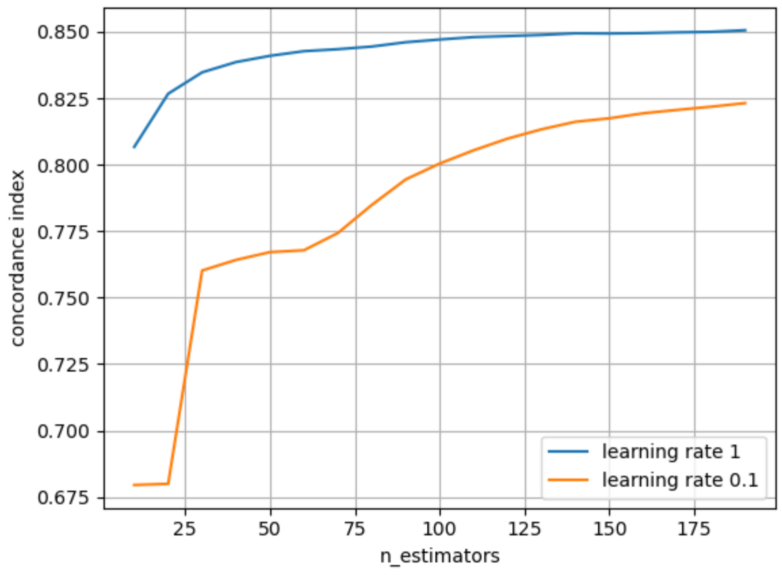
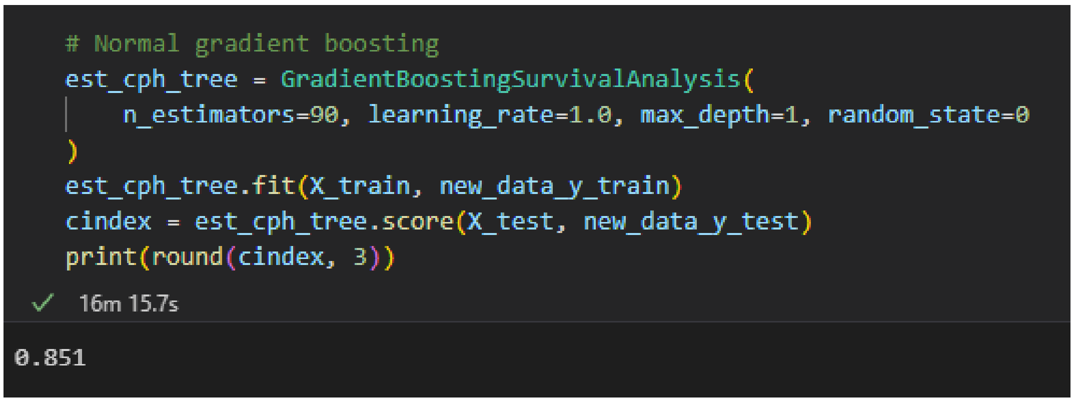


Figure 11: Component-wise parameters

We see from this output that, that increasing the learning rate, at all estimators, will increase the concordance index, thus, in the real model we should use a higher learning rate.

When implementing the normal gradient boosting machine, with the selected parameters, we achieved a concordance index of 0.843, showing that 84.3% of the patients were predicted correctly, shown here:

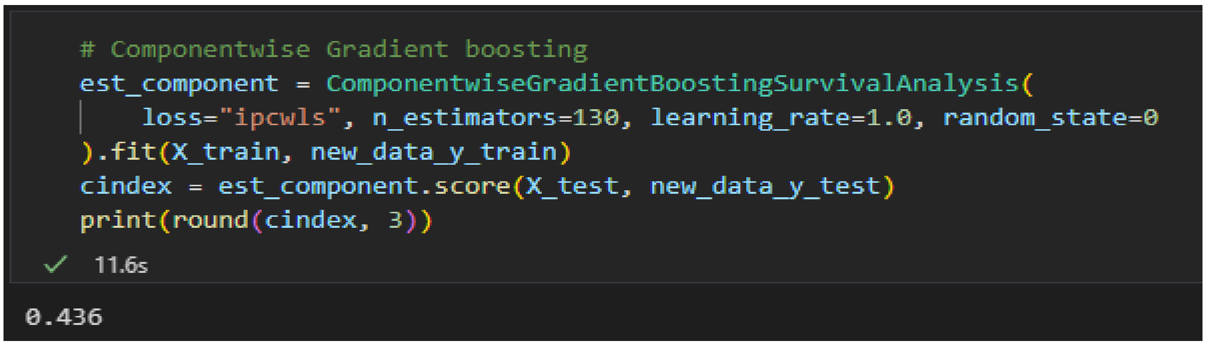
When implementing the component-wise gradient boosting machine, with the selected parameters, we achieved a concordance index of 0.436, showing that 43.6% of the patients were predicted correctly, shown here:



```
# Normal gradient boosting
est_cph_tree = GradientBoostingSurvivalAnalysis(
    n_estimators=90, learning_rate=1.0, max_depth=1, random_state=0
)
est_cph_tree.fit(X_train, new_data_y_train)
cindex = est_cph_tree.score(X_test, new_data_y_test)
print(round(cindex, 3))
```

✓ 16m 15.7s

```
0.851
```



```
# Componentwise Gradient boosting
est_component = ComponentwiseGradientBoostingSurvivalAnalysis(
    loss="ipcwls", n_estimators=130, learning_rate=1.0, random_state=0
).fit(X_train, new_data_y_train)
cindex = est_component.score(X_test, new_data_y_test)
print(round(cindex, 3))
```

```
0.436
```

As the concordance index in component-wise was significantly worse than the concordance index in normal gradient boosting machine, we rejected the component wise model and only used the normal gradient boosting machine for our survival probability plots.

We then did an importance calculation in normal model the top 5 most and least important variables were:

Most Important	Least Important
133: Platelet_amin	45: fio2_mean
25: ck_amin	44: fio2_kurtosis
100: magnesium_skew	43: fio2_amin
92: lactate_kurtosis	42: fio2_amax

We then did an importance calculation in component-wise model the top 5 most and least important variables were:

Most Important	Least Important
25: ck_amin	36: diastolic_bp_amax
28: ck_skew	100: magnesium_skew
2: alt_kurtosis	76: hb_skew
24: ck_amax	31: crp_amin

After finding out which variables were the most important in predicting which patients mortality, we then plotted the survival probability and cumulative hazard function for patients with the highest and lowest values in those important variables, comparing the difference in survival probability between the two, deriving how much that variable affects the chance of death:

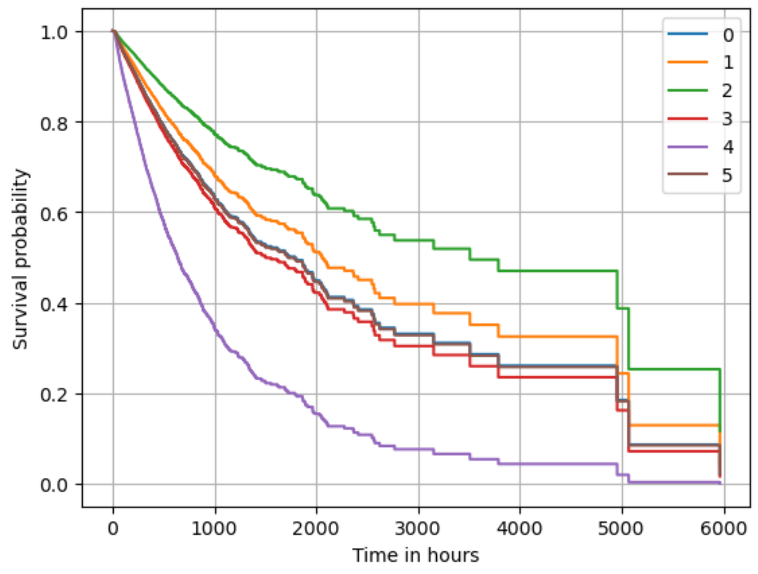


Figure 12: Most important

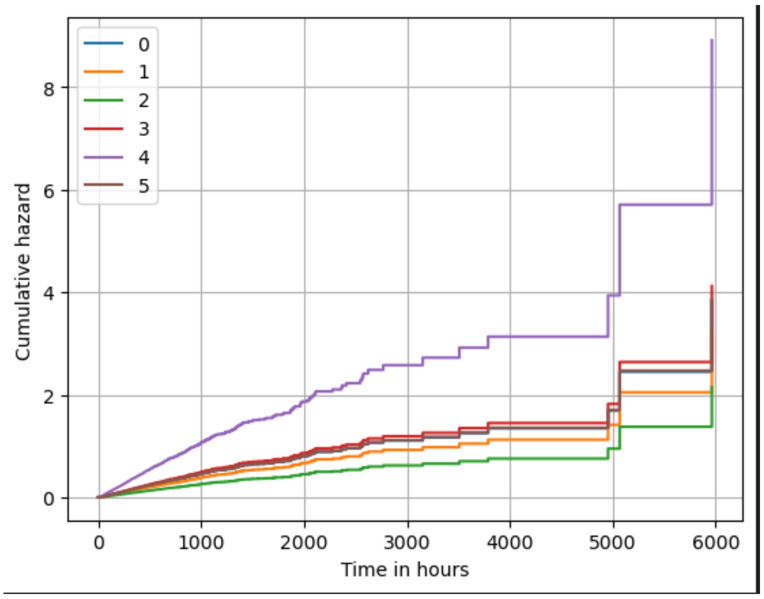


Figure 13: Least Important

In these graphs, the lines labelled 1 to 2, were the patients with the highest values in the important variables, whilst the lines labelled 3 to 5 were the patients with the least variables.

When plotting the same graphs but for the two least important variables we achieved these outputs:

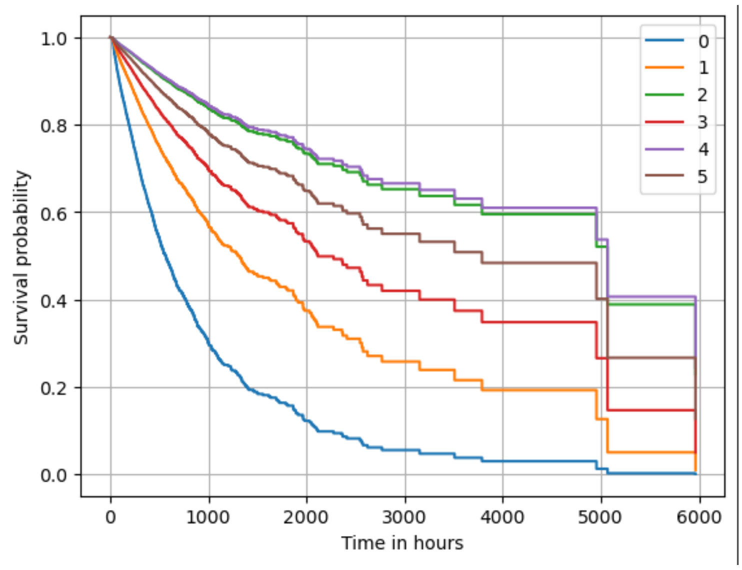


Figure 14: Most Important

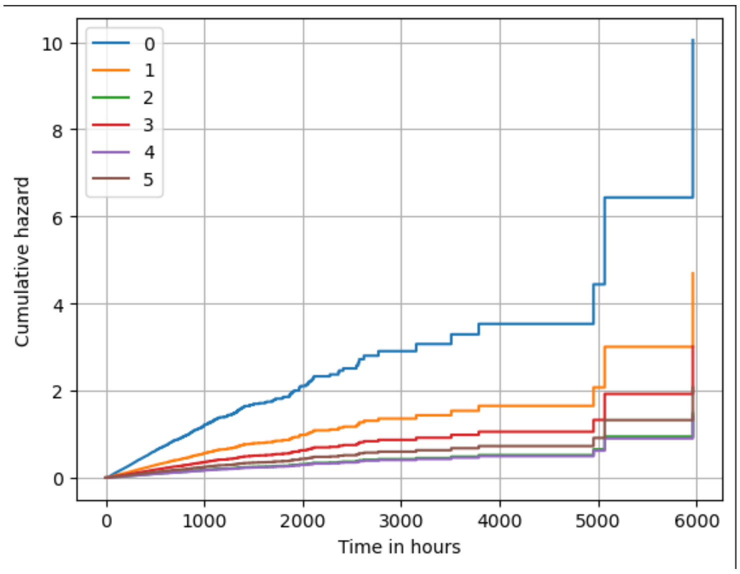


Figure 15: Least Important

Comparing the graphs of the most important variables to the least important, we see that the survival probability functions of these patients do not differ dramatically, however, the patients sampled with the most important variables did in general have a lower survival probability. The patients with more import variables, had a steeper negative gradient at the start of the study but all seemed to plateaux around the 2500 hour – 3000 hour mark.

4.4 Random Survival Forests

To repeat the method used in gradient boosted machine, the first step was to find the optimal number of estimators to use in our model.

As Random Survival Forests take significantly longer to run each time, we could not check the algorithm at as large number of estimators, in this case we checked up to 90 estimators, giving this output:

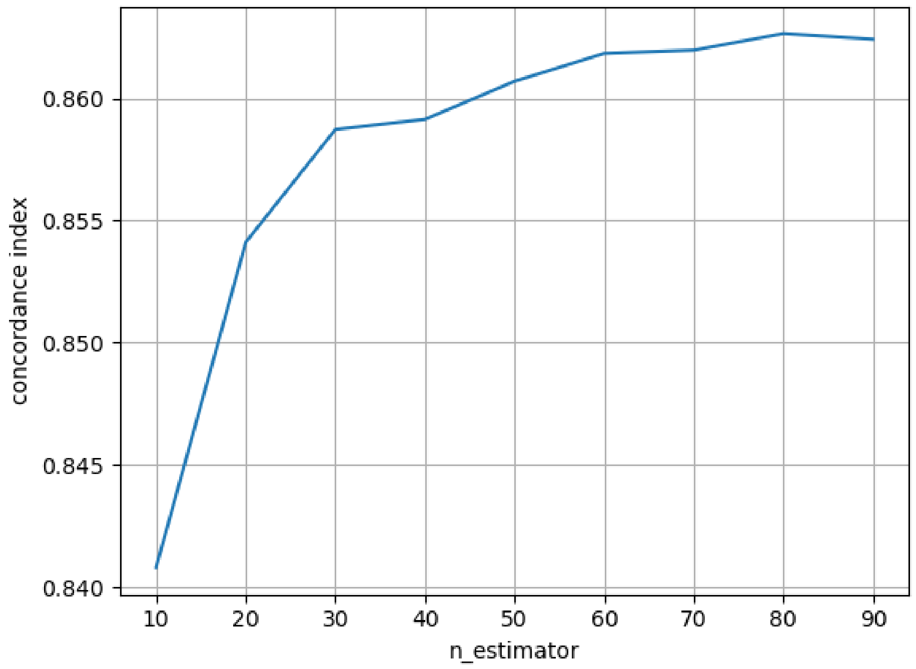


Figure 16: Random Survival Forests n_estimators

The results show that the peak concordance index occurs at 80 estimators, dropping off slightly after that point, suggesting overfitting is starting to occure. However, the difference between the concordance index of a low number of estimators, for example 10, is around 0.84 whereas the index at a high number of estimators, 90, is around 0.86, meaning that there is no major improvement. As Random Survival Forests has a much higher run time then the Gradient Boosting models, it gave more incentive to reduce the number of estimators. At the lower number of estimators, the gradient of concordance is very steep, slightly plateauing at the 30 estimator mark. This meant that choosing 30 as the number of estimators, gave the best balance between performance and computational efficiency.

To tune the other parameters, we again used the same method as before, but instead we changed the min_sample_split parameter and the min_sample_leaf parameter, keeping n_jobs and random_state the same. The model labelled 'low_sample_split' had min_sample_split = 1 and min_sample_leaf = 15. The model labelled 'low sample leaf' had min_sample_split = 10 and min_sample_leaf = 1. The model labelled 'low both' had both parameters at 1, and the model labelled 'high both' had 10 and 15 respectively. Plotting the performance of these 4 models gave:

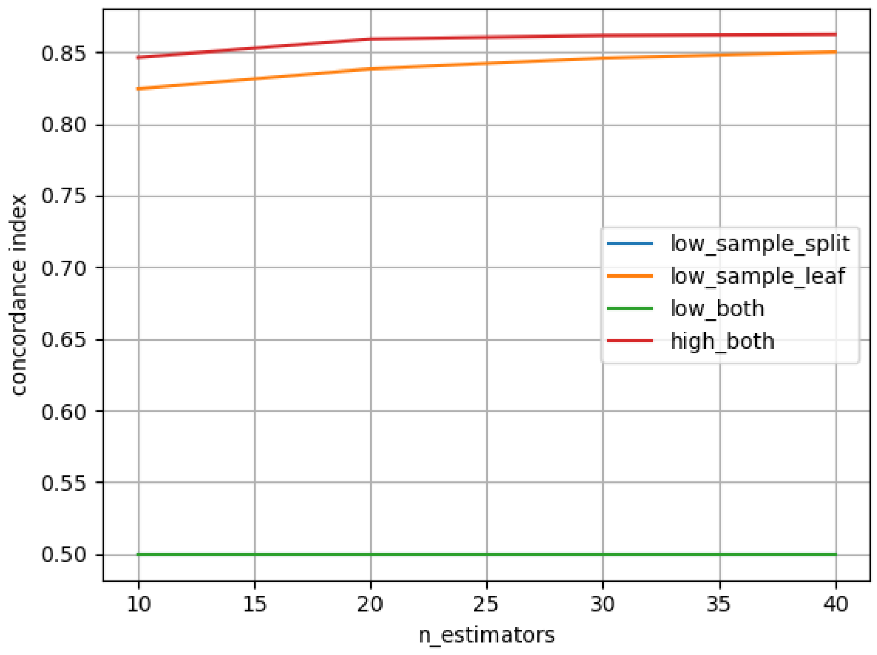
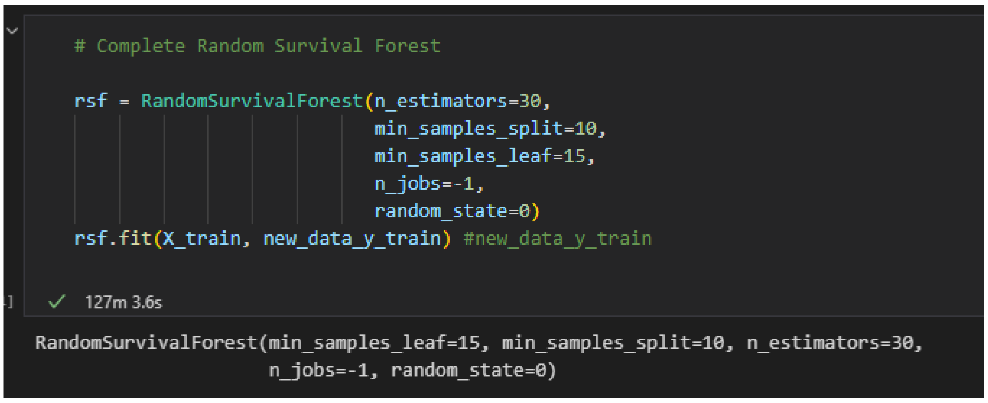


Figure 17: Random Survival Forests Parameters

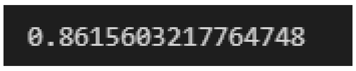
From this graph we see that the model labelled “low_both” had a concordance index of 0.5 at all number of estimators, as described before, this is an index you would expect from a random guess. The model labelled ‘high both’ had a significantly higher concordance index of around 0.85 around all number of estimators. This suggests that having a high min_sample_leaf and min_sample_split, is the best way to create our model to acquire to best performance possible.

Thus, we implemented the Random Survival Forest algorithm as follows:



```
# Complete Random Survival Forest

rsf = RandomSurvivalForest(n_estimators=30,
                           min_samples_split=10,
                           min_samples_leaf=15,
                           n_jobs=-1,
                           random_state=0)
rsf.fit(X_train, new_data_y_train) #new_data_y_train

✓ 127m 3.6s

RandomSurvivalForest(min_samples_leaf=15, min_samples_split=10, n_estimators=30,
                     n_jobs=-1, random_state=0)
```

Giving the concordance index:



```
0.8615603217764748
```

We then did an importance calculation in this model the top 5 most and least important variables were:

Most Important	Least Important
133: platelet_amin	23: capref_std
100: magnesium_skew	53: gcs_eye_std
92: lactate_kurtosis	55: gcs_motor_amin
25: ck_amin	126: phosphate_amax

Plotting the survival probability and cumulative hazard functions of the 2 most important variables gave:

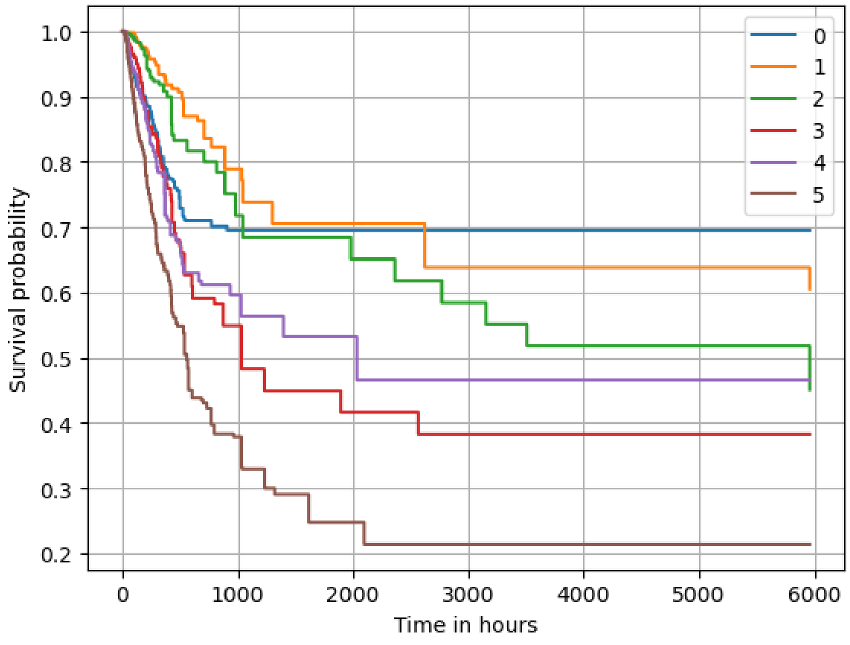


Figure 19: Most Important

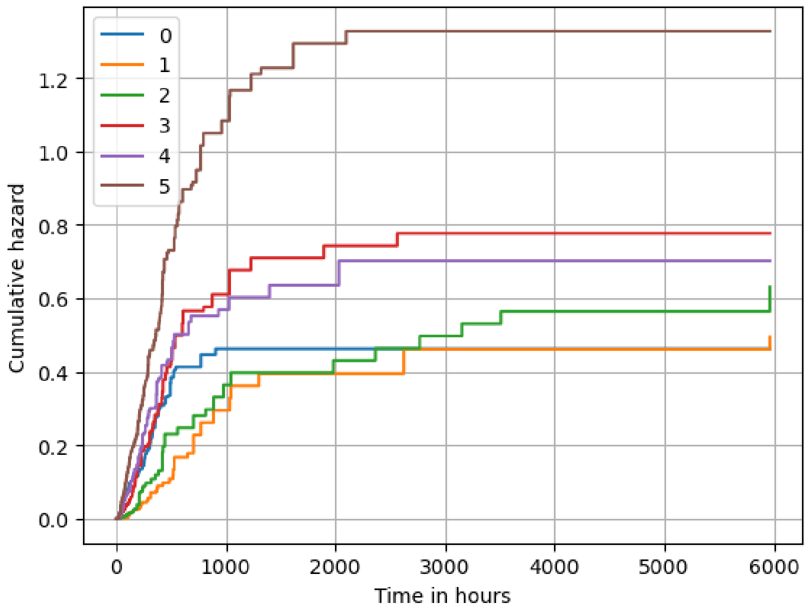


Figure 20: Least Important

Comparing to the patients sampled by least important:

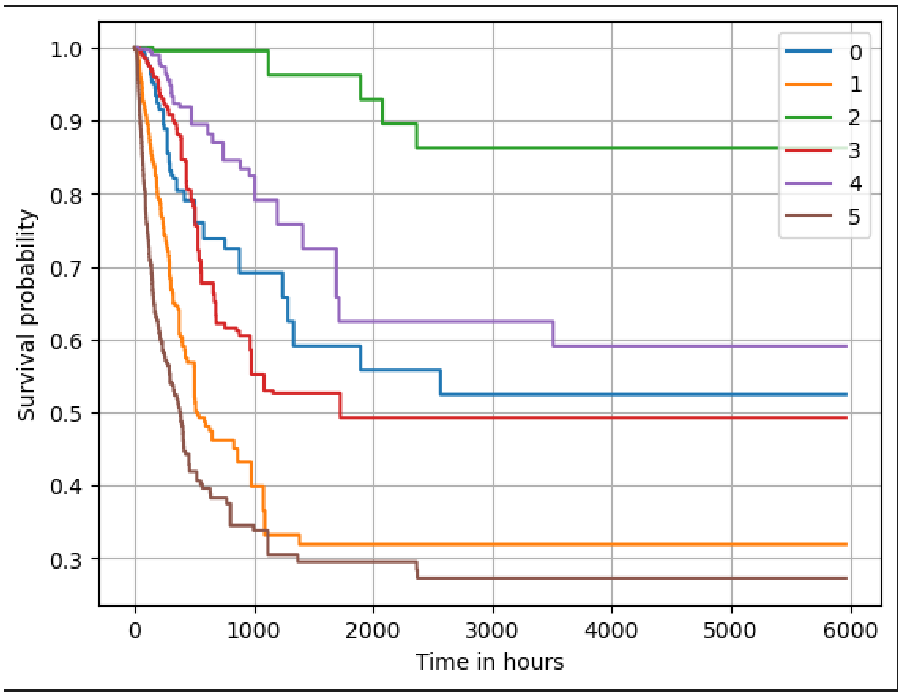


Figure 21: Most Important

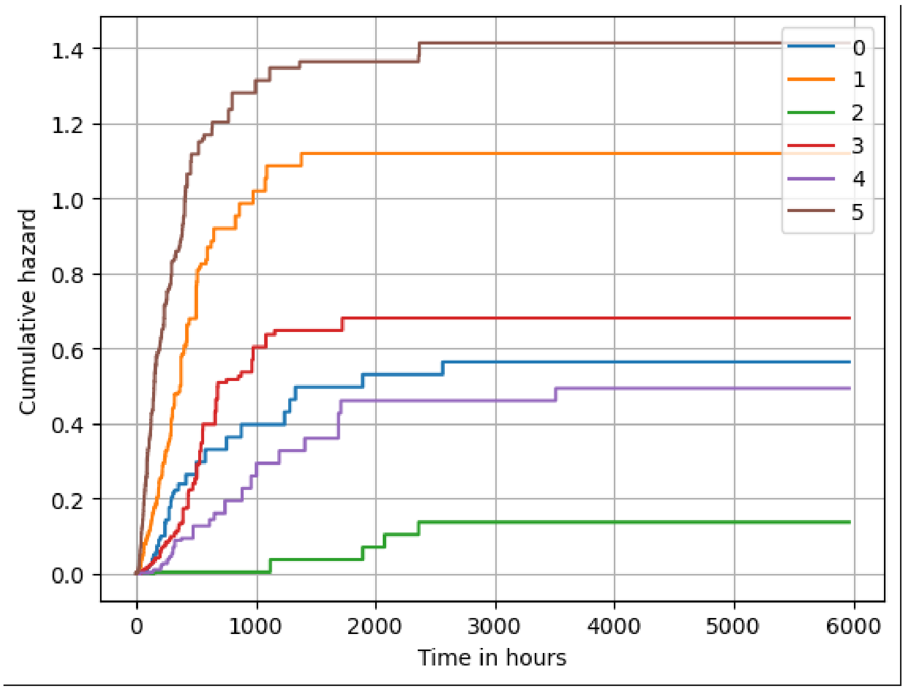


Figure 22: Least Important

Just like in Gradient Boosting machine the difference between the most important variables and the least important variables is not that large, however, the patients selected from the important variables had on average a higher cumulative hazard function, adding evidence to the validity of the model.

5 CONCLUSION

In conclusion the models ranked by performance gives

- Random Survival Forests, Concordance: 0.862
- Gradient Boosting Machine, Concordance 0.851
- Cox Regression, Concordance 0.845

The models ranked by computational efficiency:

- Cox Regression, Time <1 minute
- Gradient Boosting Machine, Time 16 minutes
- Random Survival Forests, Time 127 minutes

From our study we have found that the model with the highest performance was Random Survival Forests, showing an increase in concordance index over Gradient Boosting Machine and Cox regression by 0.011 and 0.017 respectively. This suggests to us that the best model to use when trying to predict a survival function for a patient is the Random Survival Forests. If the time increase will affect the prediction we suggest using the Gradient Boosting Machine because the performance decrease is slight.

When comparing cox regression to Gradient Boosting Machine, we find that the classical statistical to be fairly useless as Gradient Boosting Machine shows slightly more performance and both have a very small run time. Gradient Boosting Machine is also more versatile of an algorithm as it allows for tuning, meaning the concordance could be improved with more time, and cox regression cannot.

If the study could be completed again, Gradient Boosting Machine would be a good model to improve upon as when we plotted the concordance index's, we found that overfitting did not occure, meaning that the best model for Gradient Boosting Machine, was not found, thus could be improved. We could also look at the imputation of our model, perhaps imputing at the median instead of the mean could have also improved the results. The results sections could also be improved as when we compared the difference in survival function of the least important and most important vitals, we eyed to see a difference, to make this more complete and more definitive, we could complete a t test to see if the difference in functions is statistically significant.

Another way to improve the study would be to try other machine learning algorithms, even though our performance is high, other machine learning algorithms could show an improvement over even Random Survival Forests.

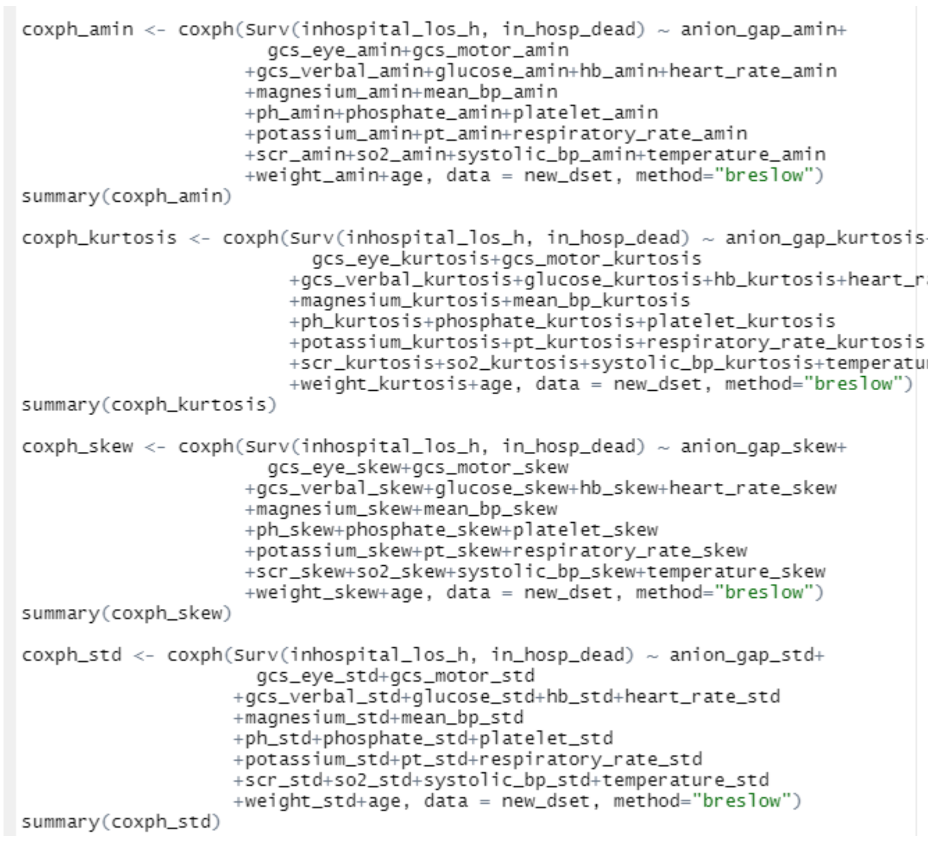
6 APPENDIX

RStudio Code

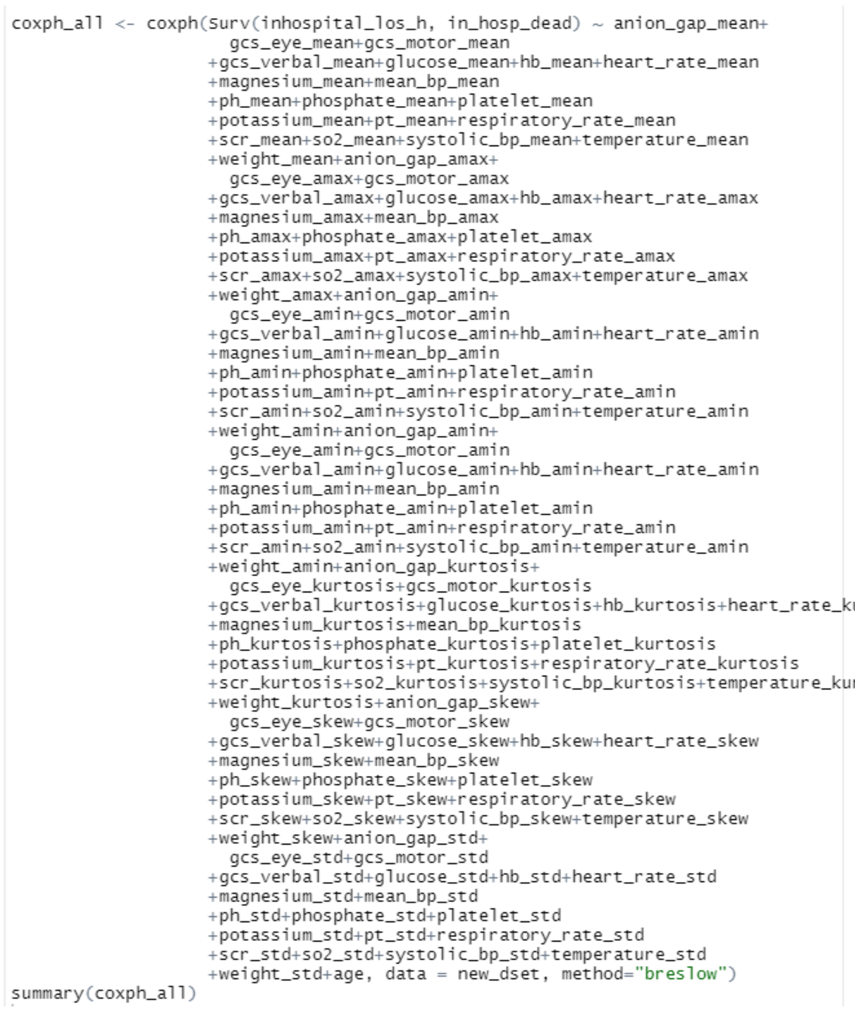
```r
library(survival)
library(ggsurvfit)
library(gtsummary)
library(tidycmprsk)
library(condSURV)
library(dplyr)
library(survminer)
library(tidyverse) # please install this package
library(mlr)

dset <- read.csv("C:/Users/Hjgra/OneDrive/Desktop/Project/data/data/events_stat_feat.csv")
head(dset)

# Check missing values
na_counts_df <- dset %>% summarise_all(~sum(is.na(.))) %>% t() %>% as.data.frame()
na_counts_df <- na_counts_df %>% rename(na_count=V1)
na_counts_df <- na_counts_df %>% mutate(na_perc=na_count/nrow(dset))

# drop columns with level of missingness > 30%
sel_cols <- na_counts_df %>% filter(na_perc<0.3)
dset <- dset %>% select(all_of(sel_cols %>% row.names()))


# impute missing values
imp <- impute(dset, classes = list(numeric=imputeMean()))
new_dset <- imp$data

########################################
# Now you can work with new_dset ######

head(new_dset[, c("inhospital_los_h", "in_hosp_dead")]) # check if the data is collected correctly

#######################################################
# Create table of surived patients and dead patients##
only_live <- data.frame(dplyr::filter(new_dset, in_hosp_dead %in% c("0")))
only_dead <- data.frame(dplyr::filter(new_dset, in_hosp_dead %in% c("1")))
```

```r
#####################################
# Keplar Meier curves ###############

survfit2(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  ggsurvfit() +
  labs(
    x = "Hours",
    y = "Overall survival probability" #Kepler Meier curve of all data
  ) +
  add_confidence_interval() +
  add_risktable()


survfit2(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset %>% filter(inhospital_los_h < 2000)) %>%
  ggsurvfit() +
  labs(
    x = "Hours",
    y = "Overall survival probability" #Kepler Meier curve of all data
  ) +
  add_confidence_interval() +
  add_risktable()

survfit2(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset %>% filter(inhospital_los_h < 4500)) %>%
  ggsurvfit() +
  labs(
    x = "Hours",
    y = "Overall survival probability" #Kepler Meier curve of all data
  ) +
  add_confidence_interval() +
  add_risktable()

survfit2(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = only_dead %>% filter(inhospital_los_h < 4500)) %>%
  ggsurvfit() +
  labs(
    x = "Hours",
    y = "Overall survival probability" #Kepler Meier curve of only people that died
  ) +
  add_confidence_interval() +
  add_risktable()

kmsurvival <- survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset)
summary(kmsurvival)

summary(survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset), times = 3000)# summary of graph
summary(survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset), times = 5200)# summary of graph
```

```r
################################################
# Summary statistics of keplar meier curves ###
survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    times = 3375,
    label_header = "**3375 hours, 75% time taken (95% CI)**"
  )

survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    times = 2250,
    label_header = "**2250 hours, 50% time taken (95% CI)**"
  )

survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    times = 1125,
    label_header = "**1125 hours, 25% time taken (95% CI)**"
  )
survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    times = 4500,
    label_header = "**4500 hours, 100% time taken (95% CI)**"
  )


survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    times = 4500,
    label_header = "**4500 hour survival (95% CI)**"
  )
survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset)

survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    probs = 0.5,
    label_header = "**Median survival (95% CI)**"
  )

survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    probs = 0.25,
    label_header = "**Lower quartile survival (95% CI)**"
  )

survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    probs = 0.75,
    label_header = "**Upper Quartile survival (95% CI)**"
  )
survfit(Surv(inhospital_los_h, in_hosp_dead) ~ 1, data = new_dset) %>%
  tbl_survfit(
    probs = 0.00,
    label_header = "**0 survival (95% CI)**"
  )

############################
# Cox regression ##########
coxph(Surv(inhospital_los_h, in_hosp_dead) ~ ethnicity, data = new_dset)
coxph(Surv(inhospital_los_h, in_hosp_dead) ~ weight_mean, data = new_dset)

coxph(Surv(inhospital_los_h, in_hosp_dead) ~ ethnicity, data = new_dset) %>%
  tbl_regression(exp = TRUE)
coxph(Surv(inhospital_los_h, in_hosp_dead) ~ weight_mean, data = new_dset) %>%
  tbl_regression(exp = TRUE)

##############################
# All means in events_stat ###
cox_means = cbind("anion_gap_mean","ca_ion_mean","ck_mean"
,"crp_mean,diastolic_bp_mean","fio2_mean","gcs_eye_mean","gcs_motor_mean"
,"gcs_verbal_mean","glucose_mean","hb_mean","heart_rate_mean"
,"height_mean","lactate_mean","magnesium_mean","mean_bp_mean"
,"o2flow_mean","peep_mean","ph_mean","phosphate_mean","platelet_mean"
,"po2_mean","potassium_mean","pt_mean","respiratory_rate_mean"
,"scr_mean","so2_mean","systolic_bp_mean","temperature_mean"
,"weight_mean,age")

################################
# means that aren't dropped #####
cox_means_select <- select(new_dset,anion_gap_mean,
                   gcs_eye_mean,gcs_motor_mean
                   ,gcs_verbal_mean,glucose_mean,hb_mean,heart_rate_mean
                   ,magnesium_mean,mean_bp_mean
                   ,ph_mean,phosphate_mean,platelet_mean
                   ,potassium_mean,pt_mean,respiratory_rate_mean
                   ,scr_mean,so2_mean,systolic_bp_mean,temperature_mean
                   ,weight_mean,age)

coxph_mean <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_mean+
               gcs_eye_mean+gcs_motor_mean
               +gcs_verbal_mean+glucose_mean+hb_mean+heart_rate_mean
               +magnesium_mean+mean_bp_mean
               +ph_mean+phosphate_mean+platelet_mean
               +potassium_mean+pt_mean+respiratory_rate_mean
               +scr_mean+so2_mean+systolic_bp_mean+temperature_mean
               +weight_mean+age, data = new_dset, method="breslow")
summary(coxph_mean)

coxph_amax <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_amax+
                      gcs_eye_amax+gcs_motor_amax
                    +gcs_verbal_amax+glucose_amax+hb_amax+heart_rate_amax
                    +magnesium_amax+mean_bp_amax
                    +ph_amax+phosphate_amax+platelet_amax
                    +potassium_amax+pt_amax+respiratory_rate_amax
                    +scr_amax+so2_amax+systolic_bp_amax+temperature_amax
                    +weight_amax+age, data = new_dset, method="breslow")
summary(coxph_amax)
```

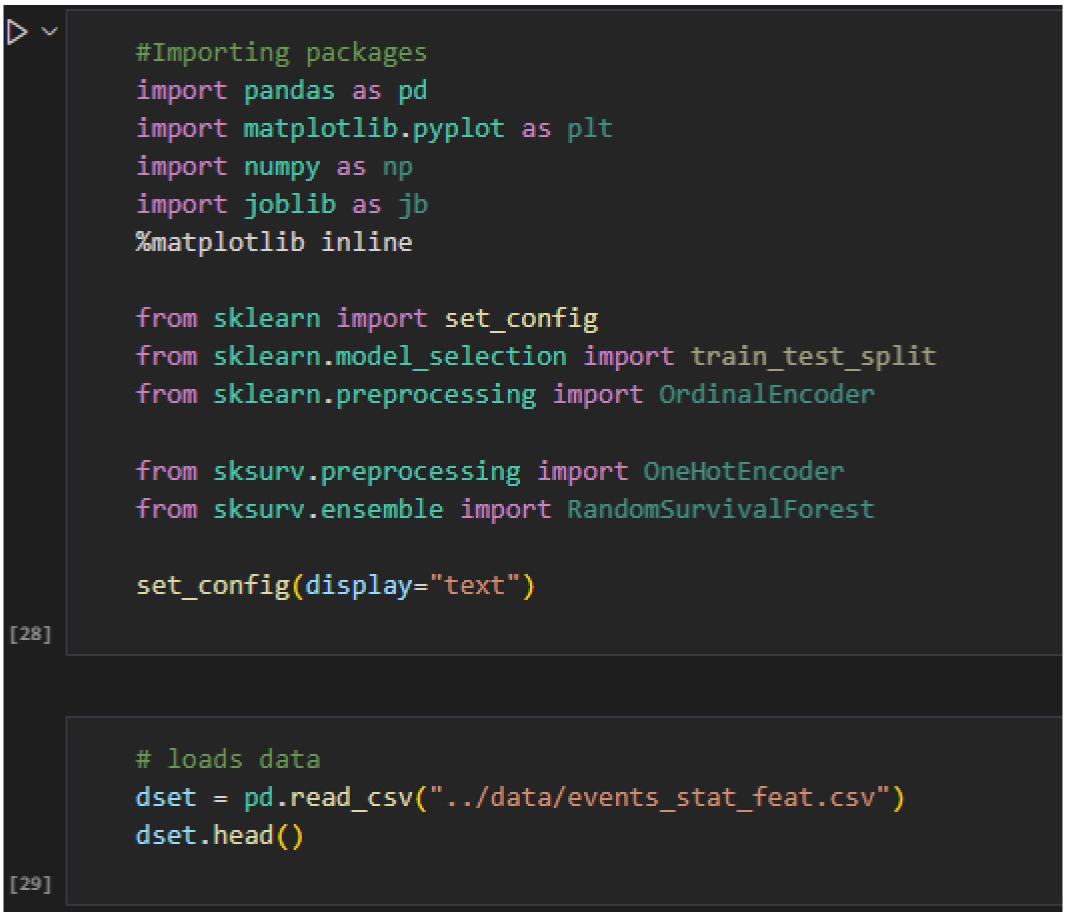


```
coxph_amin <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_amin+
                      gcs_eye_amin+gcs_motor_amin
                    +gcs_verbal_amin+glucose_amin+hb_amin+heart_rate_amin
                    +magnesium_amin+mean_bp_amin
                    +ph_amin+phosphate_amin+platelet_amin
                    +potassium_amin+pt_amin+respiratory_rate_amin
                    +scr_amin+so2_amin+systolic_bp_amin+temperature_amin
                    +weight_amin+age, data = new_dset, method="breslow")
summary(coxph_amin)

coxph_kurtosis <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_kurtosis+
                          gcs_eye_kurtosis+gcs_motor_kurtosis
                        +gcs_verbal_kurtosis+glucose_kurtosis+hb_kurtosis+heart_rate_kurtosis
                        +magnesium_kurtosis+mean_bp_kurtosis
                        +ph_kurtosis+phosphate_kurtosis+platelet_kurtosis
                        +potassium_kurtosis+pt_kurtosis+respiratory_rate_kurtosis
                        +scr_kurtosis+so2_kurtosis+systolic_bp_kurtosis+temperature_kurtosis
                        +weight_kurtosis+age, data = new_dset, method="breslow")
summary(coxph_kurtosis)

coxph_skew <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_skew+
                      gcs_eye_skew+gcs_motor_skew
                    +gcs_verbal_skew+glucose_skew+hb_skew+heart_rate_skew
                    +magnesium_skew+mean_bp_skew
                    +ph_skew+phosphate_skew+platelet_skew
                    +potassium_skew+pt_skew+respiratory_rate_skew
                    +scr_skew+so2_skew+systolic_bp_skew+temperature_skew
                    +weight_skew+age, data = new_dset, method="breslow")
summary(coxph_skew)

coxph_std <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_std+
                     gcs_eye_std+gcs_motor_std
                   +gcs_verbal_std+glucose_std+hb_std+heart_rate_std
                   +magnesium_std+mean_bp_std
                   +ph_std+phosphate_std+platelet_std
                   +potassium_std+pt_std+respiratory_rate_std
                   +scr_std+so2_std+systolic_bp_std+temperature_std
                   +weight_std+age, data = new_dset, method="breslow")
summary(coxph_std)
```



```
coxph_all <- coxph(Surv(inhospital_los_h, in_hosp_dead) ~ anion_gap_mean+
                     gcs_eye_mean+gcs_motor_mean
                   +gcs_verbal_mean+glucose_mean+hb_mean+heart_rate_mean
                   +magnesium_mean+mean_bp_mean
                   +ph_mean+phosphate_mean+platelet_mean
                   +potassium_mean+pt_mean+respiratory_rate_mean
                   +scr_mean+so2_mean+systolic_bp_mean+temperature_mean
                   +weight_mean+anion_gap_amax+
                     gcs_eye_amax+gcs_motor_amax
                   +gcs_verbal_amax+glucose_amax+hb_amax+heart_rate_amax
                   +magnesium_amax+mean_bp_amax
                   +ph_amax+phosphate_amax+platelet_amax
                   +potassium_amax+pt_amax+respiratory_rate_amax
                   +scr_amax+so2_amax+systolic_bp_amax+temperature_amax
                   +weight_amax+anion_gap_amin+
                     gcs_eye_amin+gcs_motor_amin
                   +gcs_verbal_amin+glucose_amin+hb_amin+heart_rate_amin
                   +magnesium_amin+mean_bp_amin
                   +ph_amin+phosphate_amin+platelet_amin
                   +potassium_amin+pt_amin+respiratory_rate_amin
                   +scr_amin+so2_amin+systolic_bp_amin+temperature_amin
                   +weight_amin+anion_gap_amin+
                     gcs_eye_amin+gcs_motor_amin
                   +gcs_verbal_amin+glucose_amin+hb_amin+heart_rate_amin
                   +magnesium_amin+mean_bp_amin
                   +ph_amin+phosphate_amin+platelet_amin
                   +potassium_amin+pt_amin+respiratory_rate_amin
                   +scr_amin+so2_amin+systolic_bp_amin+temperature_amin
                   +weight_amin+anion_gap_kurtosis+
                     gcs_eye_kurtosis+gcs_motor_kurtosis
                   +gcs_verbal_kurtosis+glucose_kurtosis+hb_kurtosis+heart_rate_kurtosis
                   +magnesium_kurtosis+mean_bp_kurtosis
                   +ph_kurtosis+phosphate_kurtosis+platelet_kurtosis
                   +potassium_kurtosis+pt_kurtosis+respiratory_rate_kurtosis
                   +scr_kurtosis+so2_kurtosis+systolic_bp_kurtosis+temperature_kurtosis
                   +weight_kurtosis+anion_gap_skew+
                     gcs_eye_skew+gcs_motor_skew
                   +gcs_verbal_skew+glucose_skew+hb_skew+heart_rate_skew
                   +magnesium_skew+mean_bp_skew
                   +ph_skew+phosphate_skew+platelet_skew
                   +potassium_skew+pt_skew+respiratory_rate_skew
                   +scr_skew+so2_skew+systolic_bp_skew+temperature_skew
                   +weight_skew+anion_gap_std+
                     gcs_eye_std+gcs_motor_std
                   +gcs_verbal_std+glucose_std+hb_std+heart_rate_std
                   +magnesium_std+mean_bp_std
                   +ph_std+phosphate_std+platelet_std
                   +potassium_std+pt_std+respiratory_rate_std
                   +scr_std+so2_std+systolic_bp_std+temperature_std
                   +weight_std+age, data = new_dset, method="breslow")
summary(coxph_all)
```

Python code

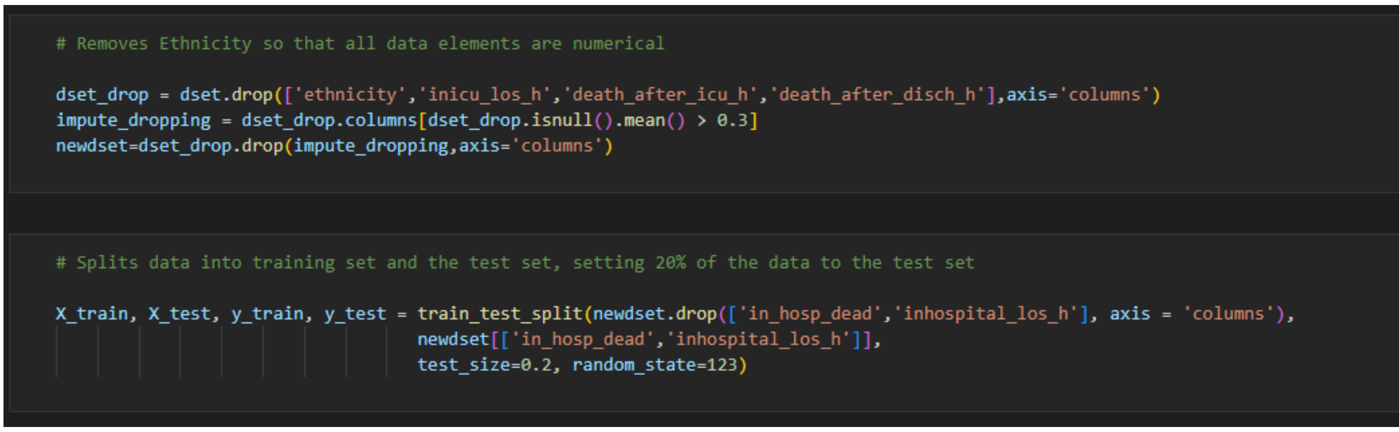


```python
#Importing packages
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import joblib as jb
%matplotlib inline

from sklearn import set_config
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder

from sksurv.preprocessing import OneHotEncoder
from sksurv.ensemble import RandomSurvivalForest

set_config(display="text")
```

[28]

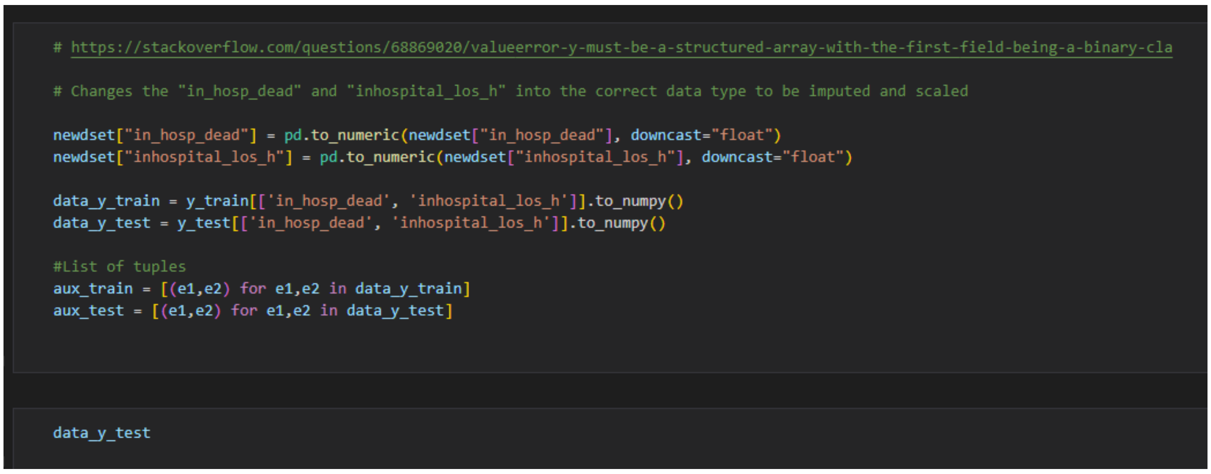
```python
# loads data
dset = pd.read_csv("../data/events_stat_feat.csv")
dset.head()
```

[29]



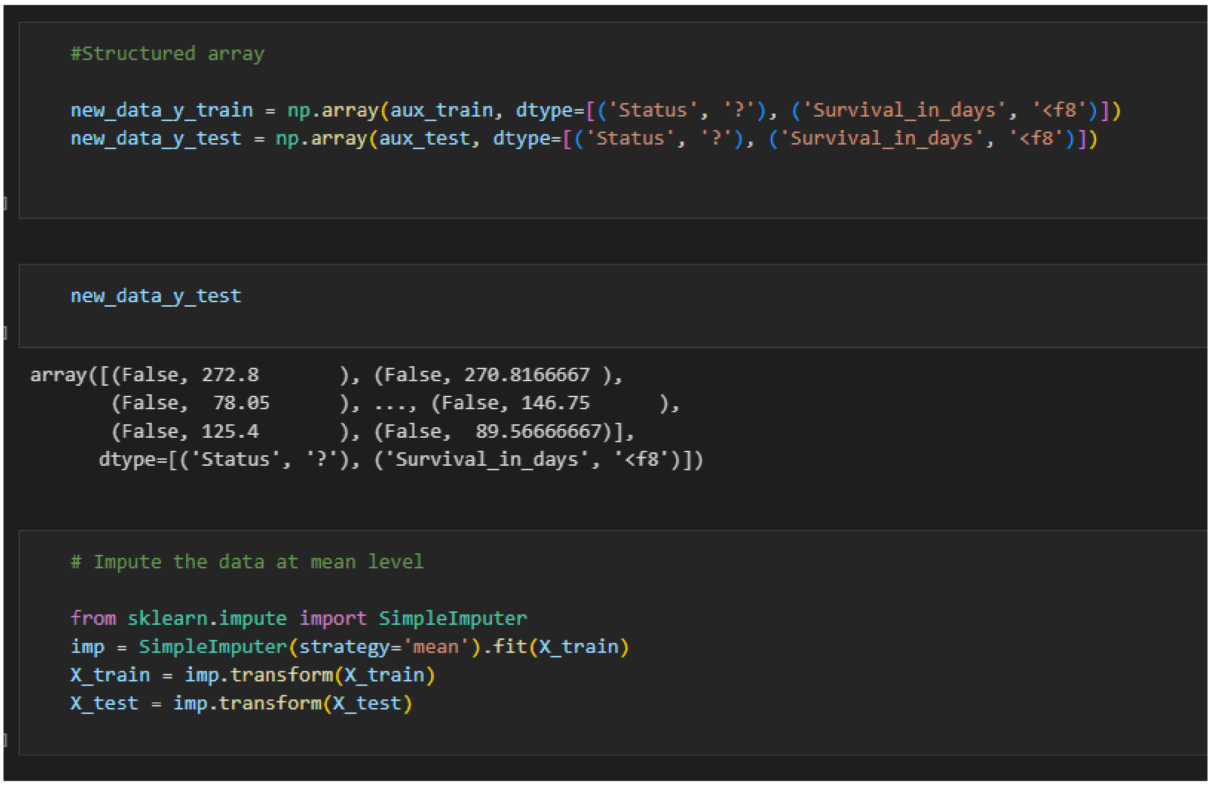
```python
# Removes Ethnicity so that all data elements are numerical

dset_drop = dset.drop(['ethnicity','inicu_los_h','death_after_icu_h','death_after_disch_h'],axis='columns')
impute_dropping = dset_drop.columns[dset_drop.isnull().mean() > 0.3]
newdset=dset_drop.drop(impute_dropping,axis='columns')
```

```python
# Splits data into training set and the test set, setting 20% of the data to the test set

X_train, X_test, y_train, y_test = train_test_split(newdset.drop(['in_hosp_dead','inhospital_los_h'], axis = 'columns'),
                                                    newdset[['in_hosp_dead','inhospital_los_h']],
                                                    test_size=0.2, random_state=123)
```



```
# https://stackoverflow.com/questions/68869020/valueerror-y-must-be-a-structured-array-with-the-first-field-being-a-binary-cla

# Changes the "in_hosp_dead" and "inhospital_los_h" into the correct data type to be imputed and scaled

newdset["in_hosp_dead"] = pd.to_numeric(newdset["in_hosp_dead"], downcast="float")
newdset["inhospital_los_h"] = pd.to_numeric(newdset["inhospital_los_h"], downcast="float")

data_y_train = y_train[['in_hosp_dead', 'inhospital_los_h']].to_numpy()
data_y_test = y_test[['in_hosp_dead', 'inhospital_los_h']].to_numpy()

#List of tuples
aux_train = [(e1,e2) for e1,e2 in data_y_train]
aux_test = [(e1,e2) for e1,e2 in data_y_test]
```

```
data_y_test
```

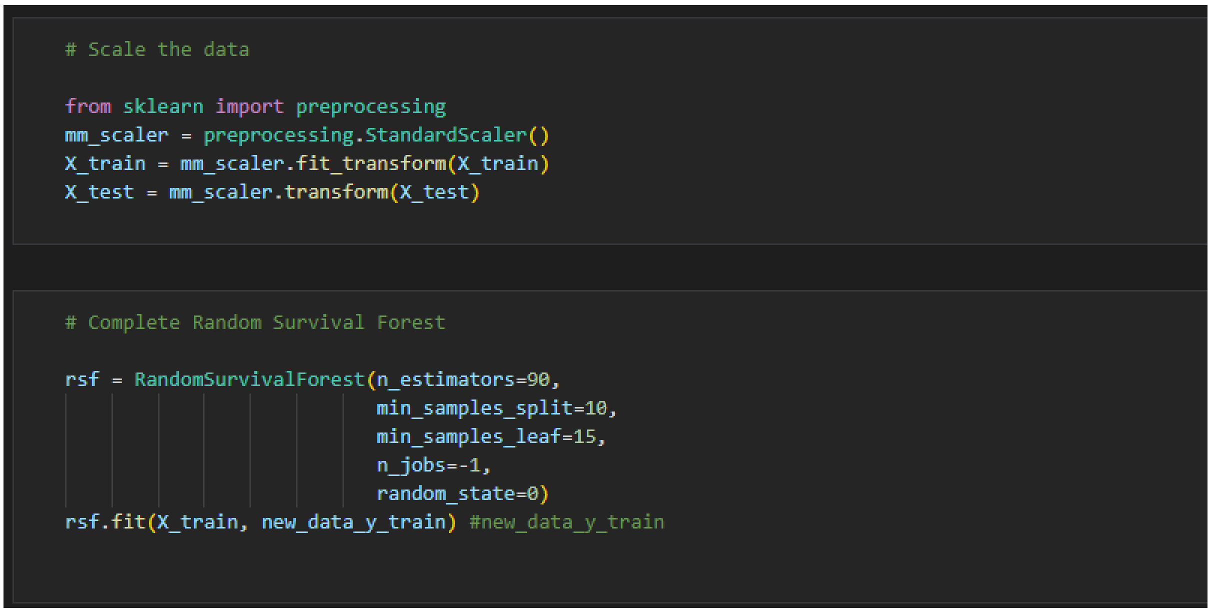


```
#Structured array

new_data_y_train = np.array(aux_train, dtype=[('Status', '?'), ('Survival_in_days', '<f8')])
new_data_y_test = np.array(aux_test, dtype=[('Status', '?'), ('Survival_in_days', '<f8')])
```

```
new_data_y_test
```

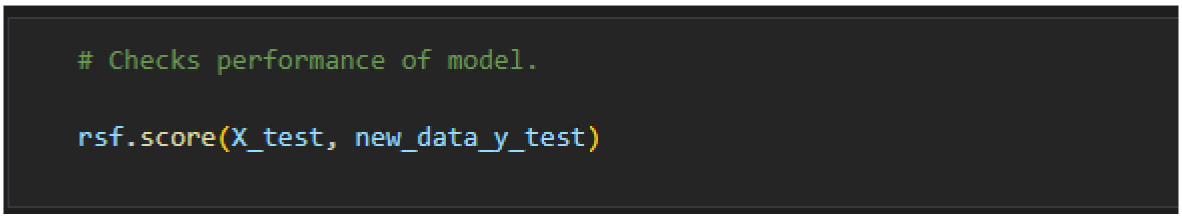
```
array([(False, 272.8       ), (False, 270.8166667 ),
       (False,  78.05      ), ..., (False, 146.75      ),
       (False, 125.4       ), (False,  89.56666667)],
      dtype=[('Status', '?'), ('Survival_in_days', '<f8')])
```

```
# Impute the data at mean level

from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean').fit(X_train)
X_train = imp.transform(X_train)
X_test = imp.transform(X_test)
```

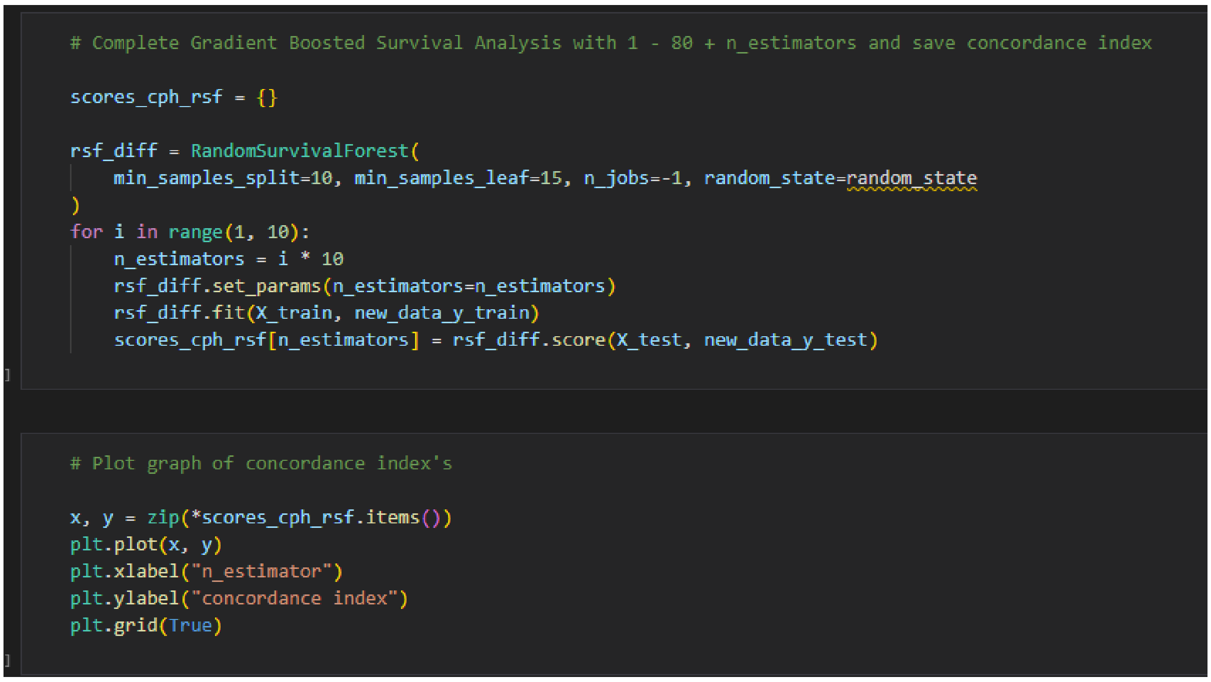


```python
# Scale the data

from sklearn import preprocessing
mm_scaler = preprocessing.StandardScaler()
X_train = mm_scaler.fit_transform(X_train)
X_test = mm_scaler.transform(X_test)
```

```python
# Complete Random Survival Forest

rsf = RandomSurvivalForest(n_estimators=90,
                           min_samples_split=10,
                           min_samples_leaf=15,
                           n_jobs=-1,
                           random_state=0)
rsf.fit(X_train, new_data_y_train) #new_data_y_train
```

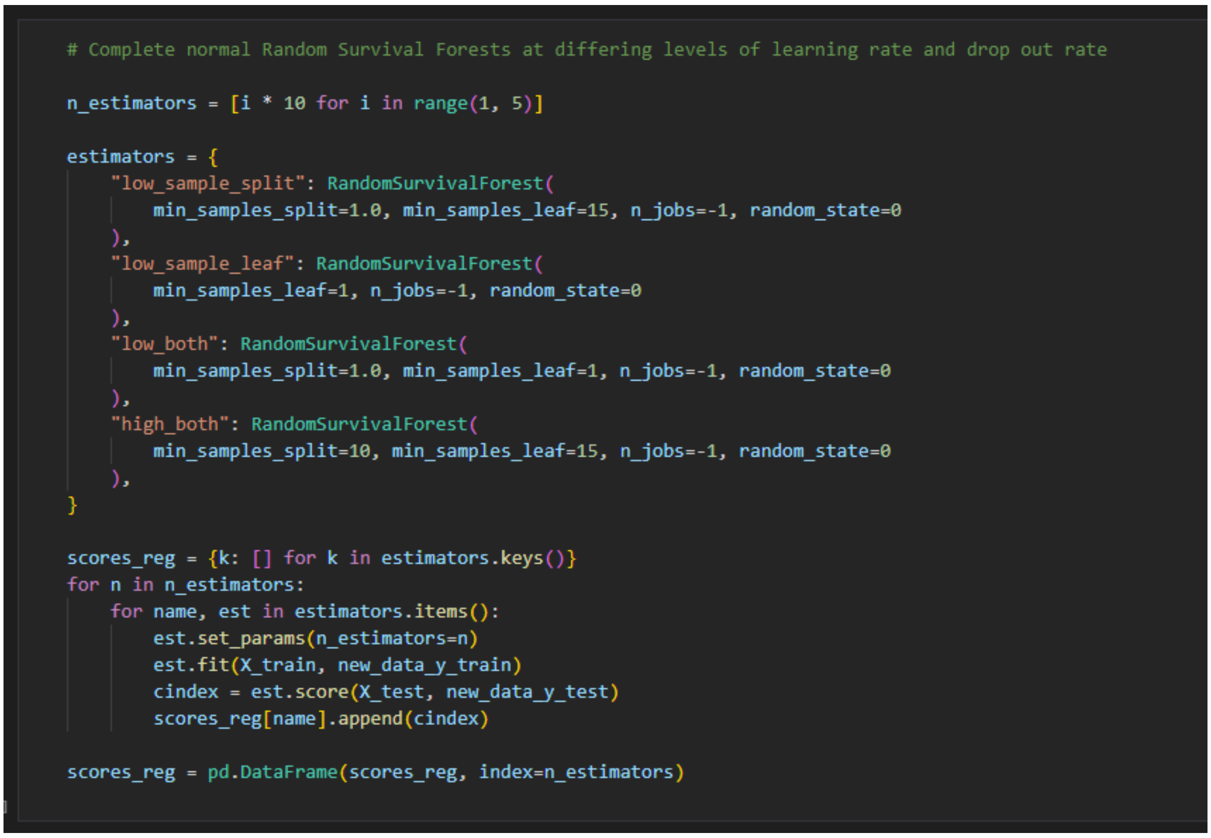


```python
# Checks performance of model.

rsf.score(X_test, new_data_y_test)
```

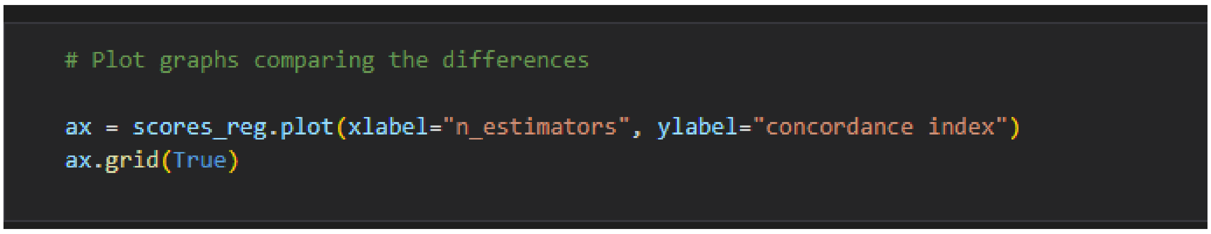


```python
# Complete Gradient Boosted Survival Analysis with 1 - 80 + n_estimators and save concordance index

scores_cph_rsf = {}

rsf_diff = RandomSurvivalForest(
    min_samples_split=10, min_samples_leaf=15, n_jobs=-1, random_state=random_state
)
for i in range(1, 10):
    n_estimators = i * 10
    rsf_diff.set_params(n_estimators=n_estimators)
    rsf_diff.fit(X_train, new_data_y_train)
    scores_cph_rsf[n_estimators] = rsf_diff.score(X_test, new_data_y_test)
```

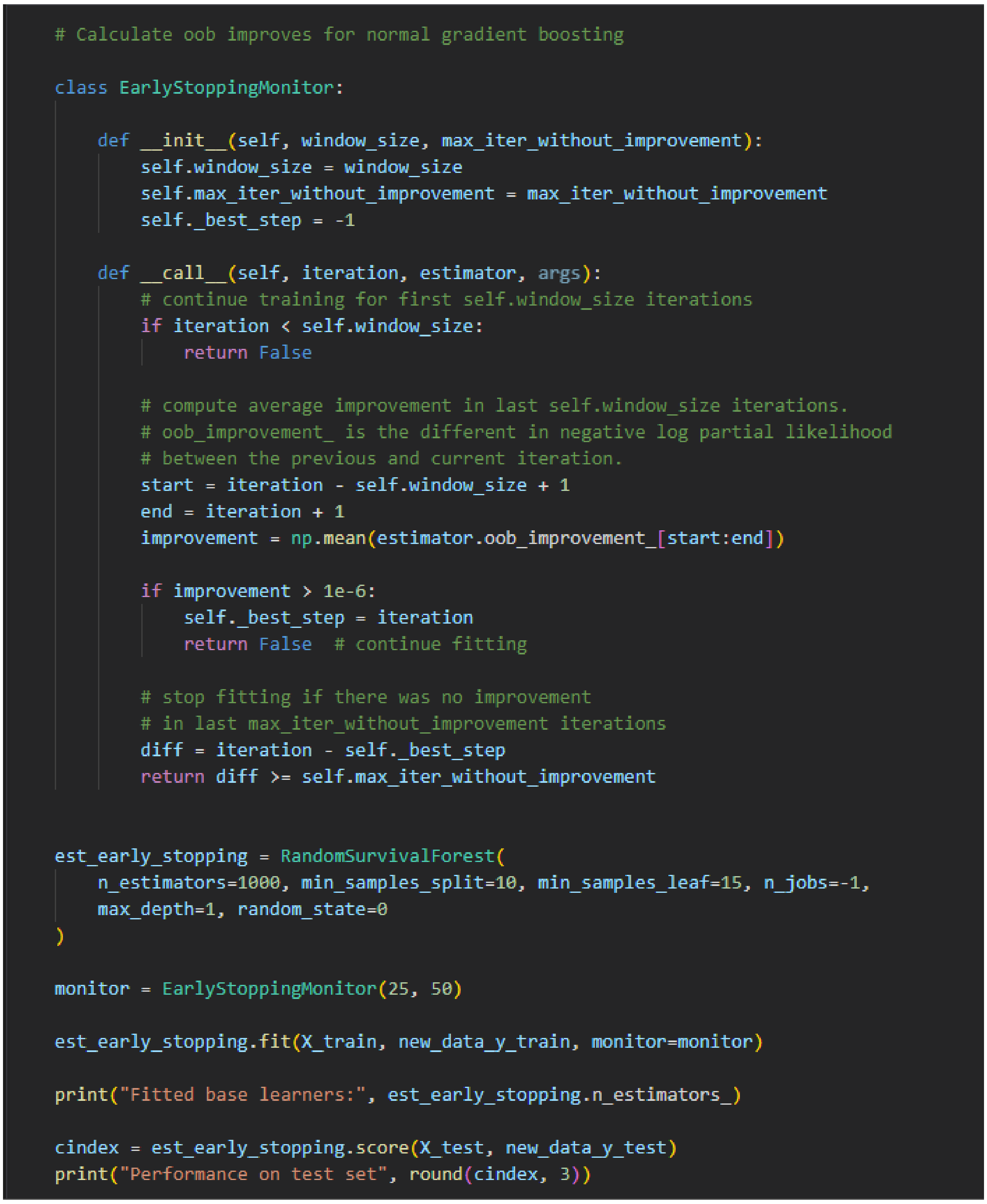
```python
# Plot graph of concordance index's

x, y = zip(*scores_cph_rsf.items())
plt.plot(x, y)
plt.xlabel("n_estimator")
plt.ylabel("concordance index")
plt.grid(True)
```



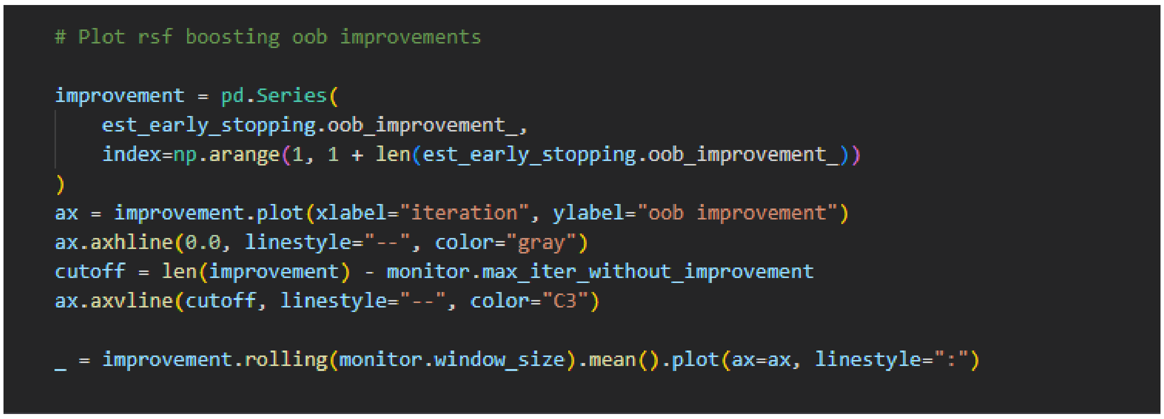
```python
# Complete normal Random Survival Forests at differing levels of learning rate and drop out rate

n_estimators = [i * 10 for i in range(1, 5)]

estimators = {
    "low_sample_split": RandomSurvivalForest(
        min_samples_split=1.0, min_samples_leaf=15, n_jobs=-1, random_state=0
    ),
    "low_sample_leaf": RandomSurvivalForest(
        min_samples_leaf=1, n_jobs=-1, random_state=0
    ),
    "low_both": RandomSurvivalForest(
        min_samples_split=1.0, min_samples_leaf=1, n_jobs=-1, random_state=0
    ),
    "high_both": RandomSurvivalForest(
        min_samples_split=10, min_samples_leaf=15, n_jobs=-1, random_state=0
    ),
}

scores_reg = {k: [] for k in estimators.keys()}
for n in n_estimators:
    for name, est in estimators.items():
        est.set_params(n_estimators=n)
        est.fit(X_train, new_data_y_train)
        cindex = est.score(X_test, new_data_y_test)
        scores_reg[name].append(cindex)

scores_reg = pd.DataFrame(scores_reg, index=n_estimators)
```



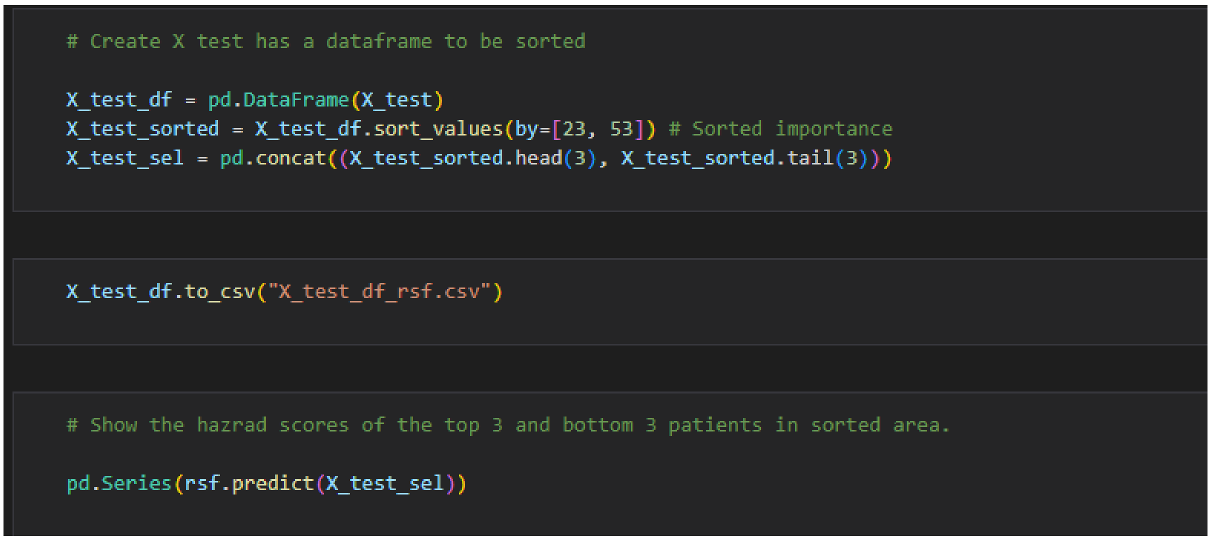
```python
# Plot graphs comparing the differences

ax = scores_reg.plot(xlabel="n_estimators", ylabel="concordance index")
ax.grid(True)
```



```python
# Calculate oob improves for normal gradient boosting

class EarlyStoppingMonitor:

    def __init__(self, window_size, max_iter_without_improvement):
        self.window_size = window_size
        self.max_iter_without_improvement = max_iter_without_improvement
        self._best_step = -1

    def __call__(self, iteration, estimator, args):
        # continue training for first self.window_size iterations
        if iteration < self.window_size:
            return False

        # compute average improvement in last self.window_size iterations.
        # oob_improvement_ is the different in negative log partial likelihood
        # between the previous and current iteration.
        start = iteration - self.window_size + 1
        end = iteration + 1
        improvement = np.mean(estimator.oob_improvement_[start:end])

        if improvement > 1e-6:
            self._best_step = iteration
            return False  # continue fitting

        # stop fitting if there was no improvement
        # in last max_iter_without_improvement iterations
        diff = iteration - self._best_step
        return diff >= self.max_iter_without_improvement


est_early_stopping = RandomSurvivalForest(
    n_estimators=1000, min_samples_split=10, min_samples_leaf=15, n_jobs=-1,
    max_depth=1, random_state=0
)

monitor = EarlyStoppingMonitor(25, 50)

est_early_stopping.fit(X_train, new_data_y_train, monitor=monitor)

print("Fitted base learners:", est_early_stopping.n_estimators_)

cindex = est_early_stopping.score(X_test, new_data_y_test)
print("Performance on test set", round(cindex, 3))
```



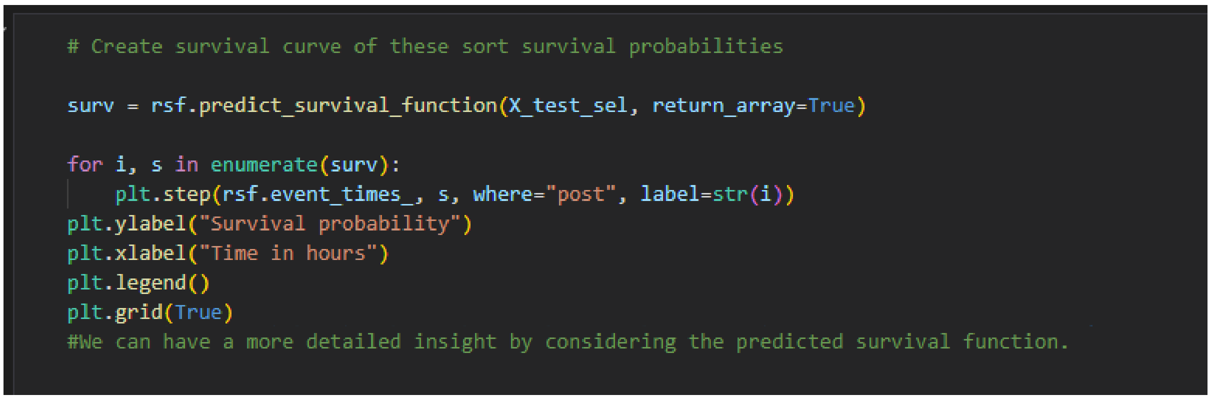
```python
# Plot rsf boosting oob improvements

improvement = pd.Series(
    est_early_stopping.oob_improvement_,
    index=np.arange(1, 1 + len(est_early_stopping.oob_improvement_))
)
ax = improvement.plot(xlabel="iteration", ylabel="oob improvement")
ax.axhline(0.0, linestyle="--", color="gray")
cutoff = len(improvement) - monitor.max_iter_without_improvement
ax.axvline(cutoff, linestyle="--", color="C3")

_ = improvement.rolling(monitor.window_size).mean().plot(ax=ax, linestyle=":")
```



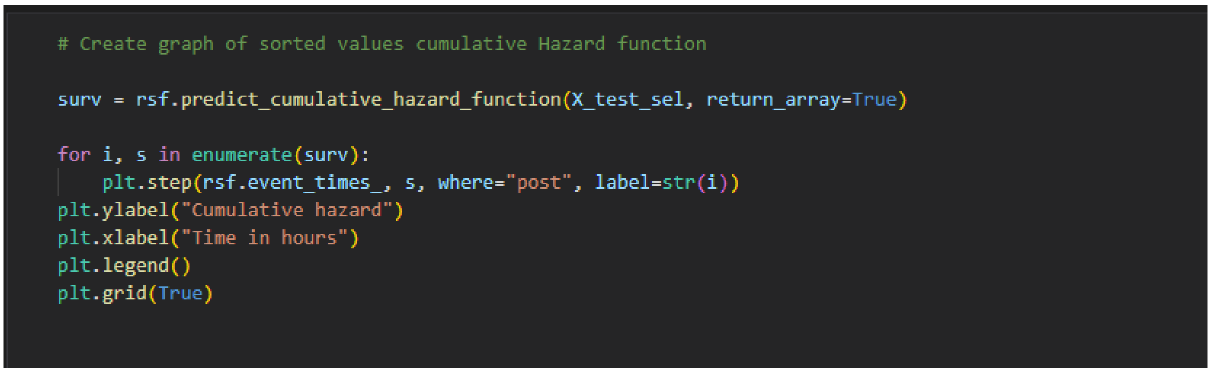
```python
# Create X test has a dataframe to be sorted

X_test_df = pd.DataFrame(X_test)
X_test_sorted = X_test_df.sort_values(by=[23, 53]) # Sorted importance
X_test_sel = pd.concat((X_test_sorted.head(3), X_test_sorted.tail(3)))
```

```python
X_test_df.to_csv("X_test_df_rsf.csv")
```

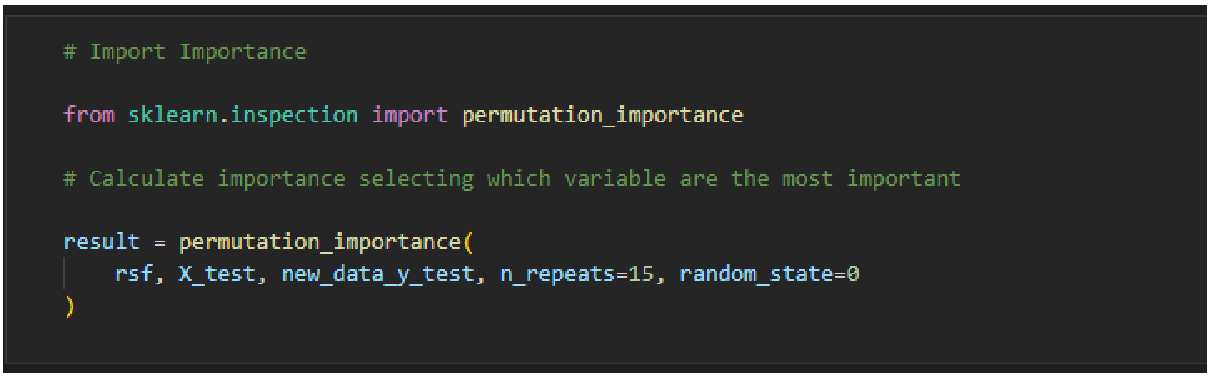
```python
# Show the hazrad scores of the top 3 and bottom 3 patients in sorted area.

pd.Series(rsf.predict(X_test_sel))
```



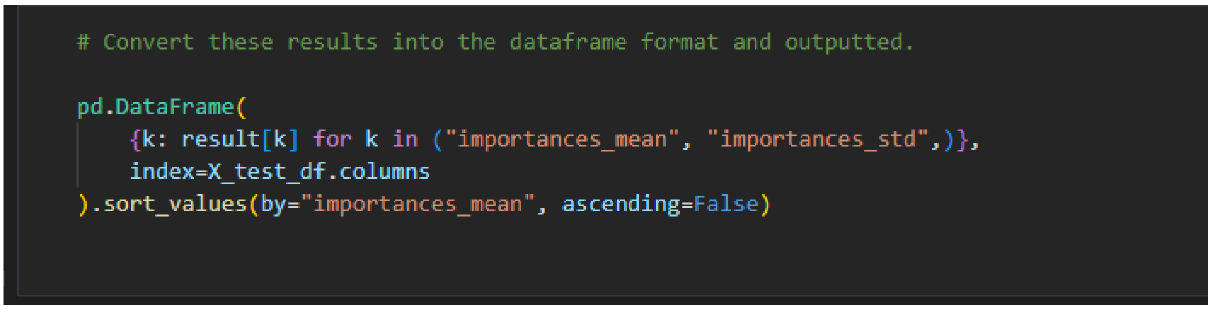
```python
# Create survival curve of these sort survival probabilities

surv = rsf.predict_survival_function(X_test_sel, return_array=True)

for i, s in enumerate(surv):
    plt.step(rsf.event_times_, s, where="post", label=str(i))
plt.ylabel("Survival probability")
plt.xlabel("Time in hours")
plt.legend()
plt.grid(True)
#We can have a more detailed insight by considering the predicted survival function.
```



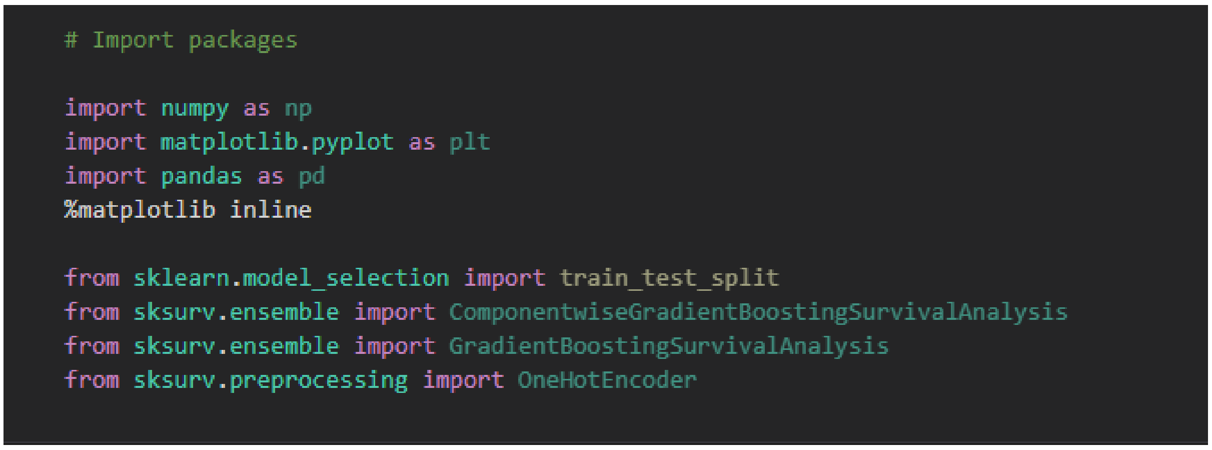
```python
# Create graph of sorted values cumulative Hazard function

surv = rsf.predict_cumulative_hazard_function(X_test_sel, return_array=True)

for i, s in enumerate(surv):
    plt.step(rsf.event_times_, s, where="post", label=str(i))
plt.ylabel("Cumulative hazard")
plt.xlabel("Time in hours")
plt.legend()
plt.grid(True)
```



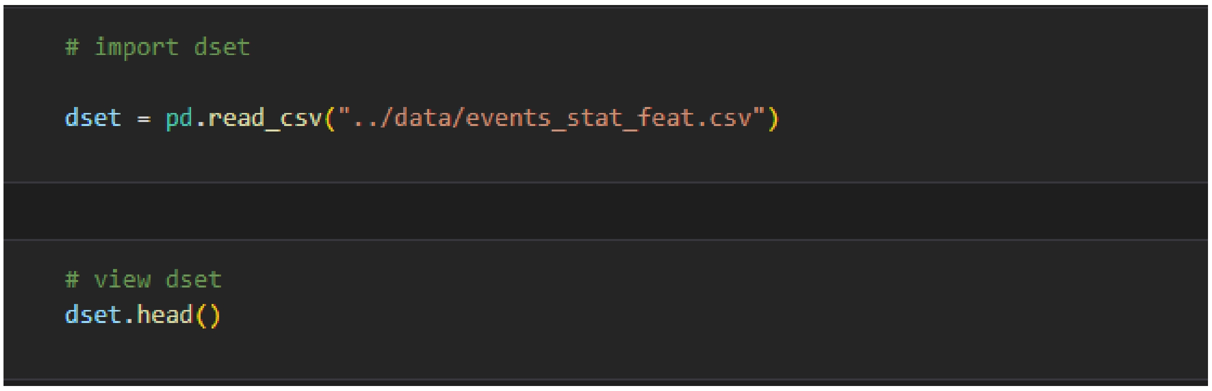
```python
# Import Importance

from sklearn.inspection import permutation_importance

# Calculate importance selecting which variable are the most important

result = permutation_importance(
    rsf, X_test, new_data_y_test, n_repeats=15, random_state=0
)
```



```
# Convert these results into the dataframe format and outputted.

pd.DataFrame(
    {k: result[k] for k in ("importances_mean", "importances_std",)},
    index=X_test_df.columns
).sort_values(by="importances_mean", ascending=False)
```

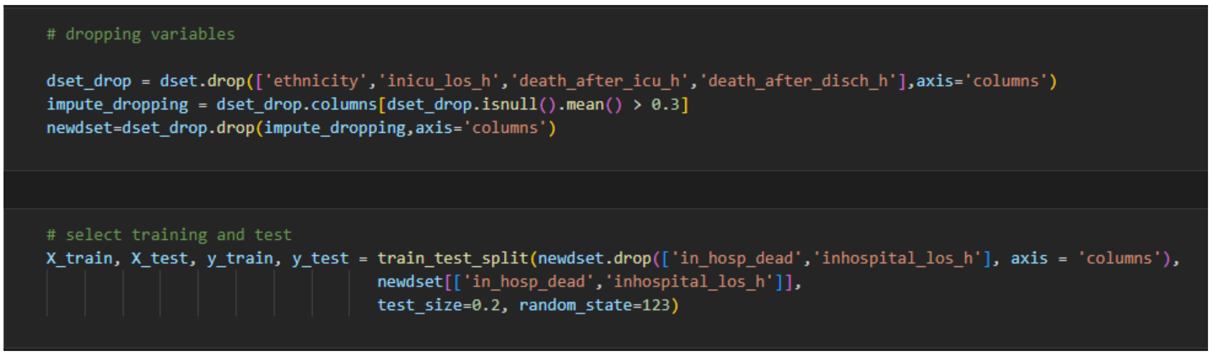


```
# Import packages

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

from sklearn.model_selection import train_test_split
from sksurv.ensemble import ComponentwiseGradientBoostingSurvivalAnalysis
from sksurv.ensemble import GradientBoostingSurvivalAnalysis
from sksurv.preprocessing import OneHotEncoder
```



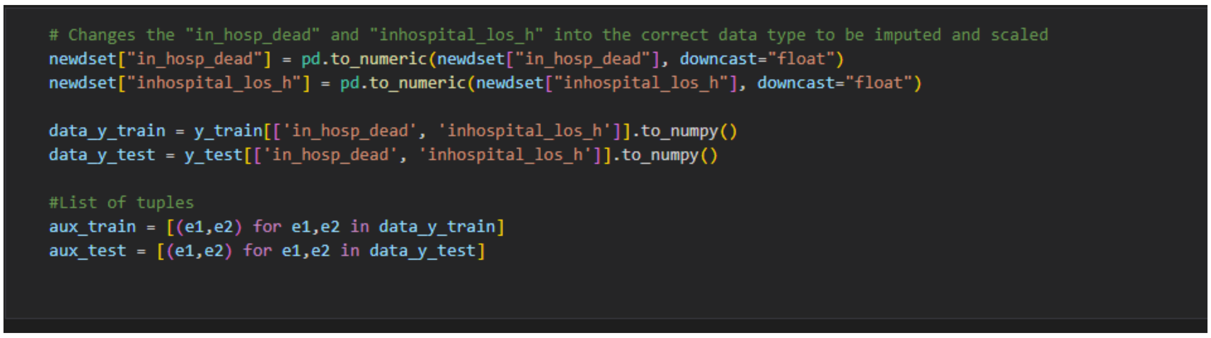
```
# import dset

dset = pd.read_csv("../data/events_stat_feat.csv")
```

```
# view dset
dset.head()
```

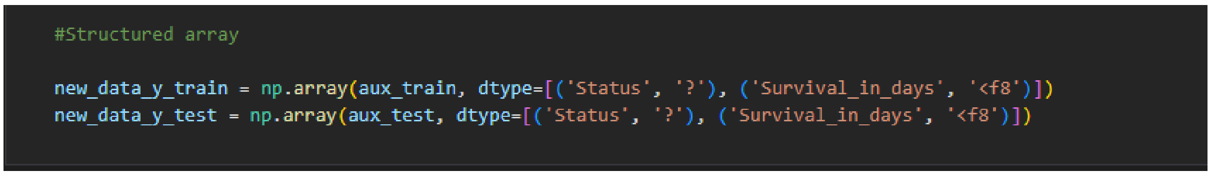


```
# dropping variables

dset_drop = dset.drop(['ethnicity','inicu_los_h','death_after_icu_h','death_after_disch_h'],axis='columns')
impute_dropping = dset_drop.columns[dset_drop.isnull().mean() > 0.3]
newdset=dset_drop.drop(impute_dropping,axis='columns')
```

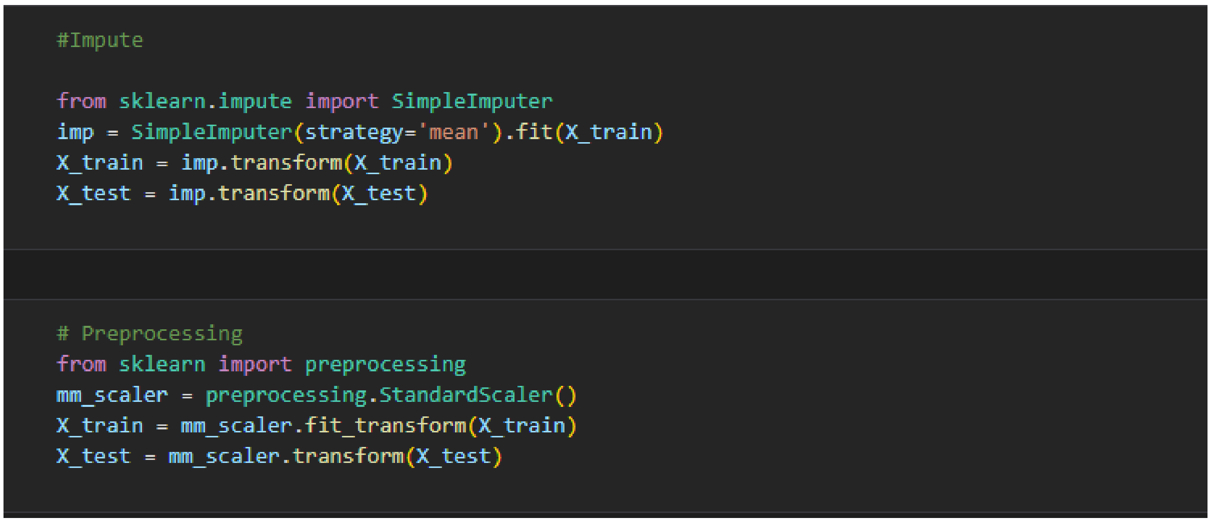
```
# select training and test
X_train, X_test, y_train, y_test = train_test_split(newdset.drop(['in_hosp_dead','inhospital_los_h'], axis = 'columns'),
                                                    newdset[['in_hosp_dead','inhospital_los_h']],
                                                    test_size=0.2, random_state=123)
```



```python
# Changes the "in_hosp_dead" and "inhospital_los_h" into the correct data type to be imputed and scaled
newdset["in_hosp_dead"] = pd.to_numeric(newdset["in_hosp_dead"], downcast="float")
newdset["inhospital_los_h"] = pd.to_numeric(newdset["inhospital_los_h"], downcast="float")

data_y_train = y_train[['in_hosp_dead', 'inhospital_los_h']].to_numpy()
data_y_test = y_test[['in_hosp_dead', 'inhospital_los_h']].to_numpy()

#List of tuples
aux_train = [(e1,e2) for e1,e2 in data_y_train]
aux_test = [(e1,e2) for e1,e2 in data_y_test]
```

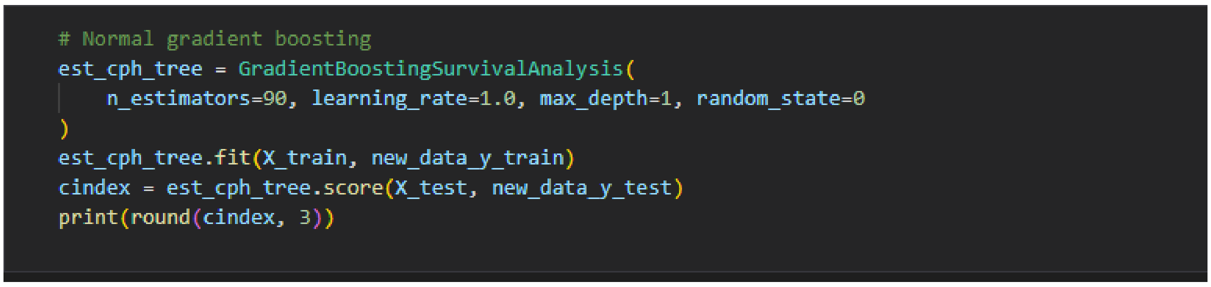


```python
#Structured array

new_data_y_train = np.array(aux_train, dtype=[('Status', '?'), ('Survival_in_days', '<f8')])
new_data_y_test = np.array(aux_test, dtype=[('Status', '?'), ('Survival_in_days', '<f8')])
```

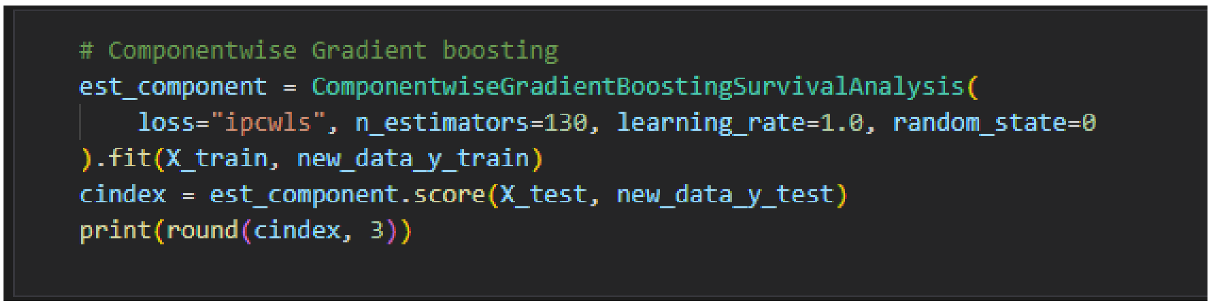


```python
#Impute

from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean').fit(X_train)
X_train = imp.transform(X_train)
X_test = imp.transform(X_test)
```

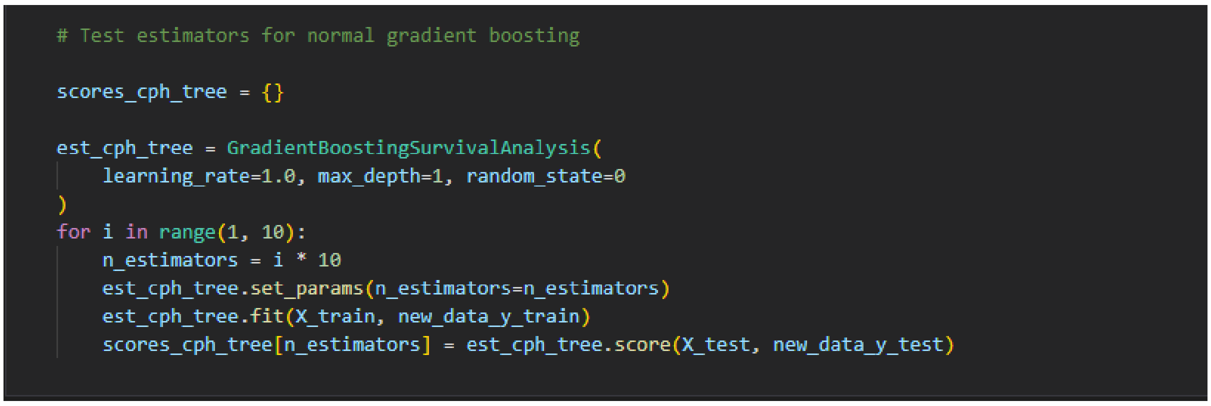
```python
# Preprocessing
from sklearn import preprocessing
mm_scaler = preprocessing.StandardScaler()
X_train = mm_scaler.fit_transform(X_train)
X_test = mm_scaler.transform(X_test)
```



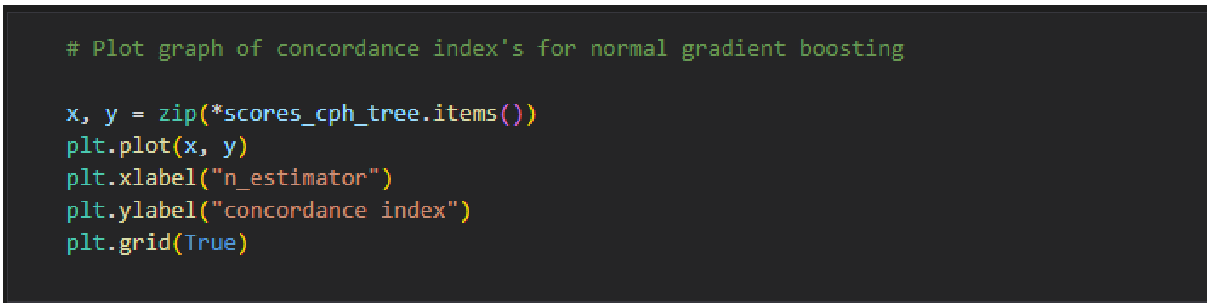
```python
# Normal gradient boosting
est_cph_tree = GradientBoostingSurvivalAnalysis(
    n_estimators=90, learning_rate=1.0, max_depth=1, random_state=0
)
est_cph_tree.fit(X_train, new_data_y_train)
cindex = est_cph_tree.score(X_test, new_data_y_test)
print(round(cindex, 3))
```



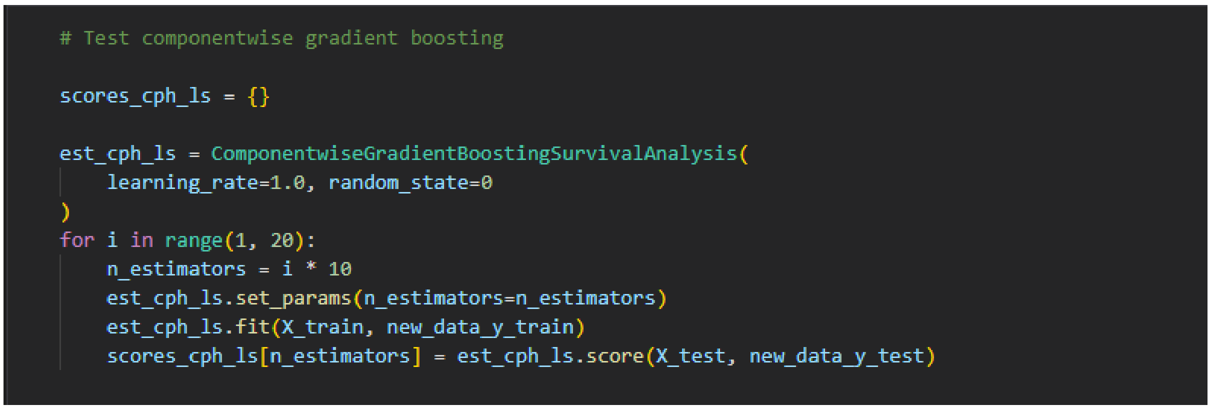
```python
# Componentwise Gradient boosting
est_component = ComponentwiseGradientBoostingSurvivalAnalysis(
    loss="ipcwls", n_estimators=130, learning_rate=1.0, random_state=0
).fit(X_train, new_data_y_train)
cindex = est_component.score(X_test, new_data_y_test)
print(round(cindex, 3))
```



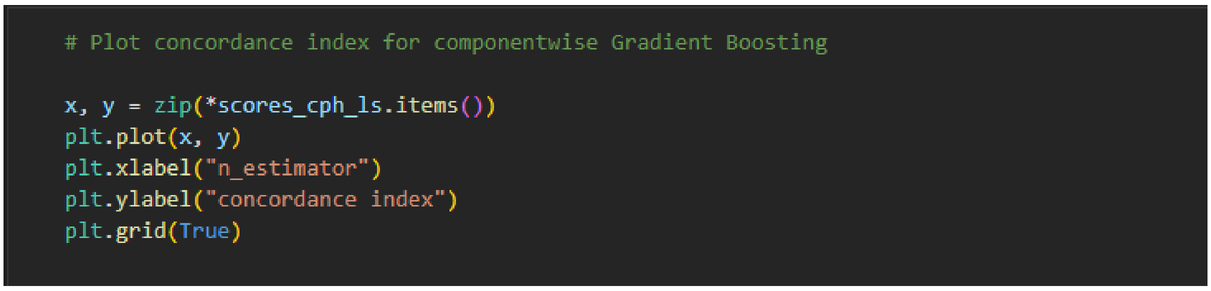
```python
# Test estimators for normal gradient boosting

scores_cph_tree = {}

est_cph_tree = GradientBoostingSurvivalAnalysis(
    learning_rate=1.0, max_depth=1, random_state=0
)
for i in range(1, 10):
    n_estimators = i * 10
    est_cph_tree.set_params(n_estimators=n_estimators)
    est_cph_tree.fit(X_train, new_data_y_train)
    scores_cph_tree[n_estimators] = est_cph_tree.score(X_test, new_data_y_test)
```



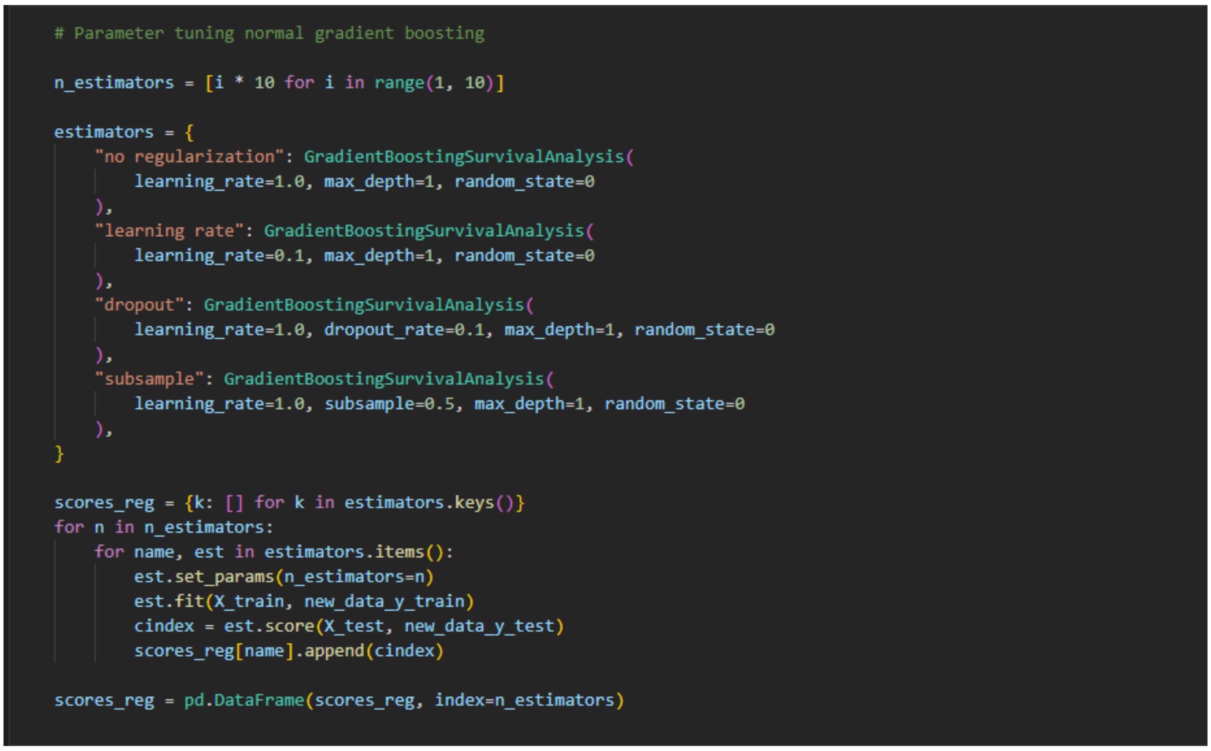
```python
# Plot graph of concordance index's for normal gradient boosting

x, y = zip(*scores_cph_tree.items())
plt.plot(x, y)
plt.xlabel("n_estimator")
plt.ylabel("concordance index")
plt.grid(True)
```



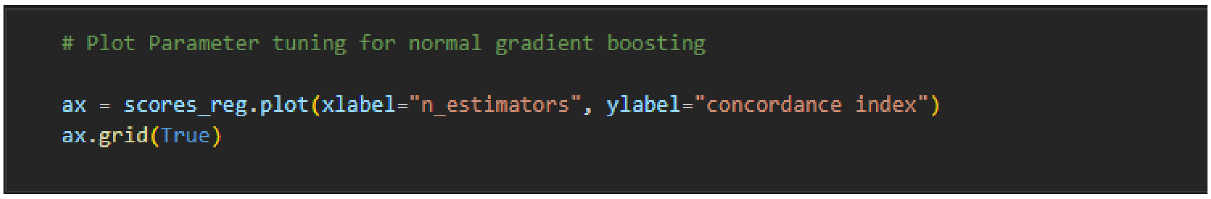
```python
# Test componentwise gradient boosting

scores_cph_ls = {}

est_cph_ls = ComponentwiseGradientBoostingSurvivalAnalysis(
    learning_rate=1.0, random_state=0
)
for i in range(1, 20):
    n_estimators = i * 10
    est_cph_ls.set_params(n_estimators=n_estimators)
    est_cph_ls.fit(X_train, new_data_y_train)
    scores_cph_ls[n_estimators] = est_cph_ls.score(X_test, new_data_y_test)
```



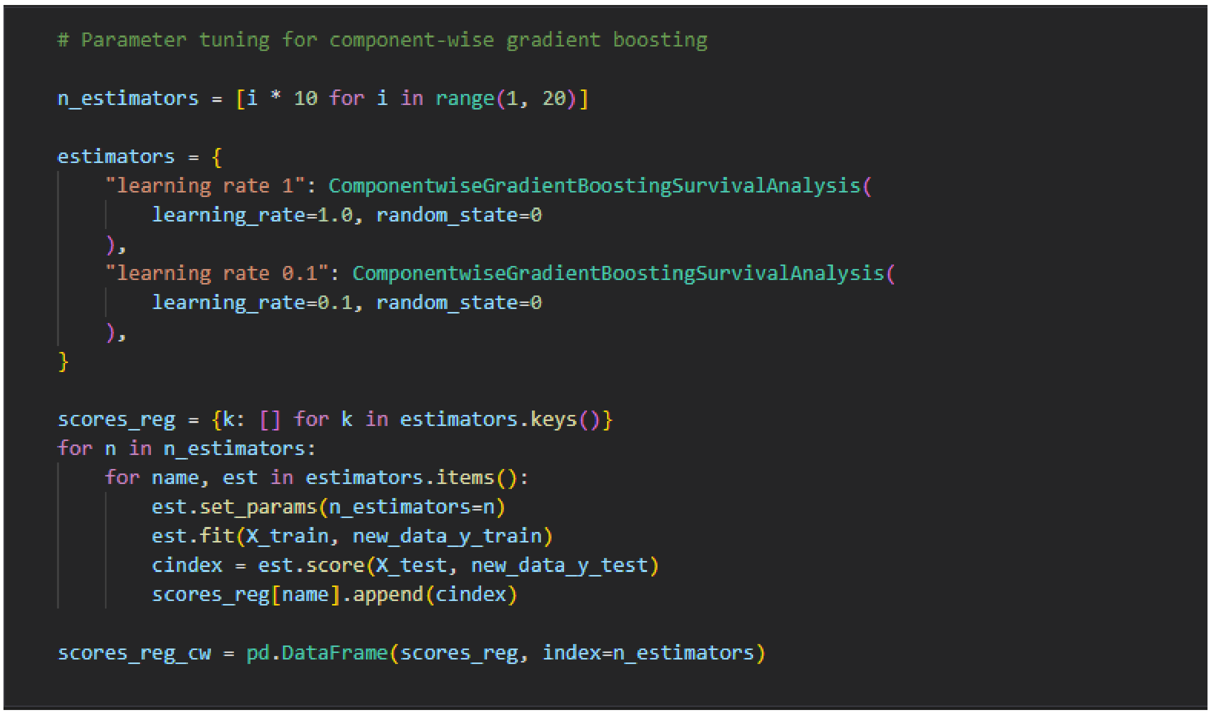
```python
# Plot concordance index for componentwise Gradient Boosting

x, y = zip(*scores_cph_ls.items())
plt.plot(x, y)
plt.xlabel("n_estimator")
plt.ylabel("concordance index")
plt.grid(True)
```



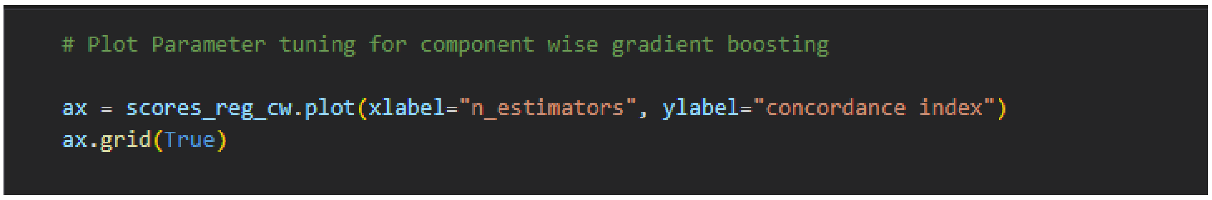
```python
# Parameter tuning normal gradient boosting

n_estimators = [i * 10 for i in range(1, 10)]

estimators = {
    "no regularization": GradientBoostingSurvivalAnalysis(
        learning_rate=1.0, max_depth=1, random_state=0
    ),
    "learning rate": GradientBoostingSurvivalAnalysis(
        learning_rate=0.1, max_depth=1, random_state=0
    ),
    "dropout": GradientBoostingSurvivalAnalysis(
        learning_rate=1.0, dropout_rate=0.1, max_depth=1, random_state=0
    ),
    "subsample": GradientBoostingSurvivalAnalysis(
        learning_rate=1.0, subsample=0.5, max_depth=1, random_state=0
    ),
}

scores_reg = {k: [] for k in estimators.keys()}
for n in n_estimators:
    for name, est in estimators.items():
        est.set_params(n_estimators=n)
        est.fit(X_train, new_data_y_train)
        cindex = est.score(X_test, new_data_y_test)
        scores_reg[name].append(cindex)

scores_reg = pd.DataFrame(scores_reg, index=n_estimators)
```



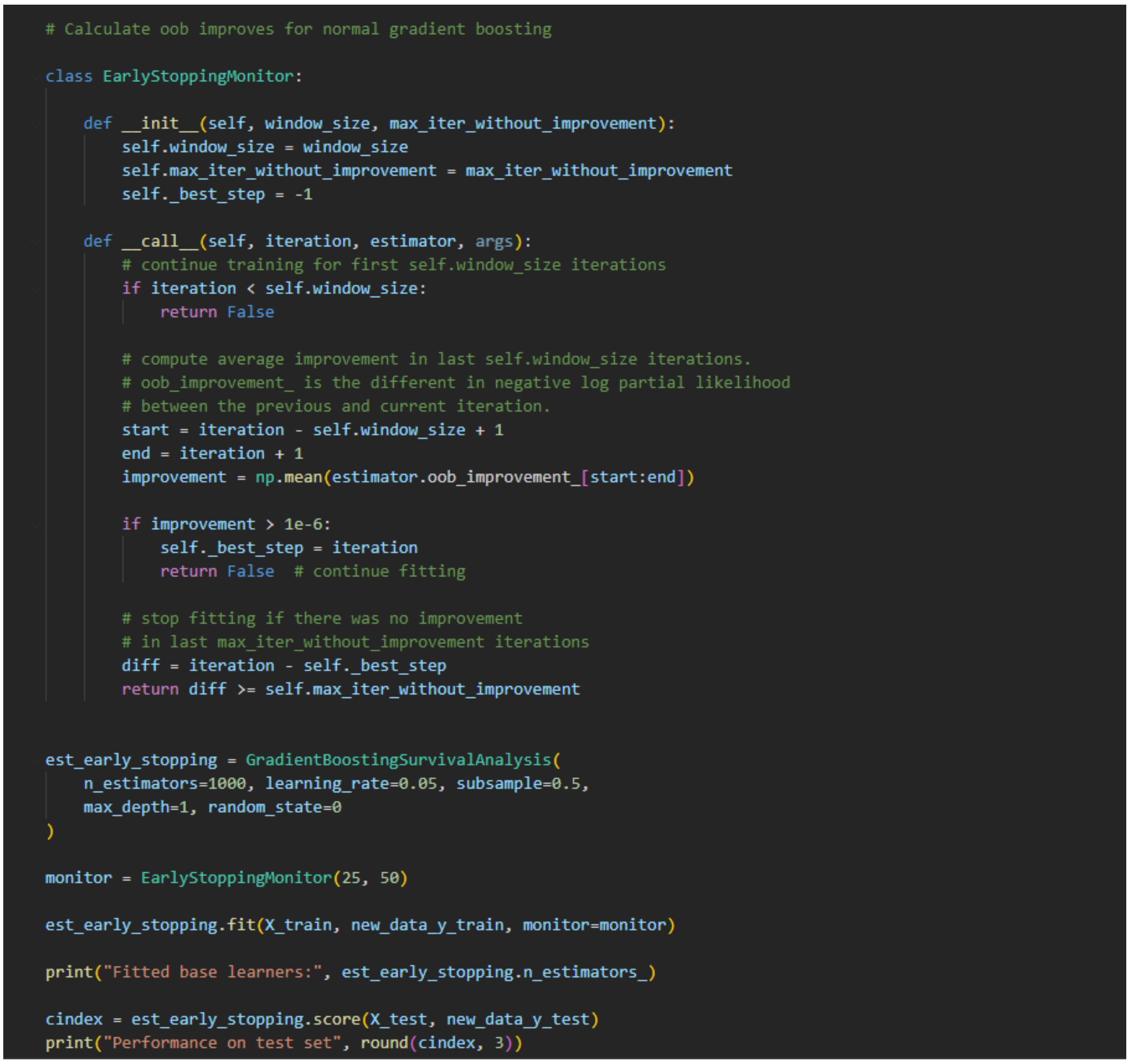
```python
# Plot Parameter tuning for normal gradient boosting

ax = scores_reg.plot(xlabel="n_estimators", ylabel="concordance index")
ax.grid(True)
```



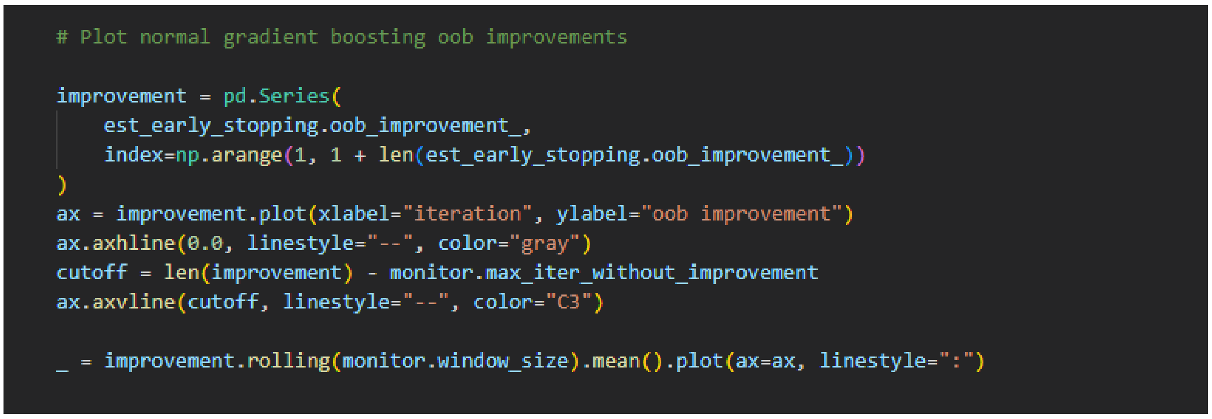
```python
# Parameter tuning for component-wise gradient boosting

n_estimators = [i * 10 for i in range(1, 20)]

estimators = {
    "learning rate 1": ComponentwiseGradientBoostingSurvivalAnalysis(
        learning_rate=1.0, random_state=0
    ),
    "learning rate 0.1": ComponentwiseGradientBoostingSurvivalAnalysis(
        learning_rate=0.1, random_state=0
    ),
}

scores_reg = {k: [] for k in estimators.keys()}
for n in n_estimators:
    for name, est in estimators.items():
        est.set_params(n_estimators=n)
        est.fit(X_train, new_data_y_train)
        cindex = est.score(X_test, new_data_y_test)
        scores_reg[name].append(cindex)

scores_reg_cw = pd.DataFrame(scores_reg, index=n_estimators)
```



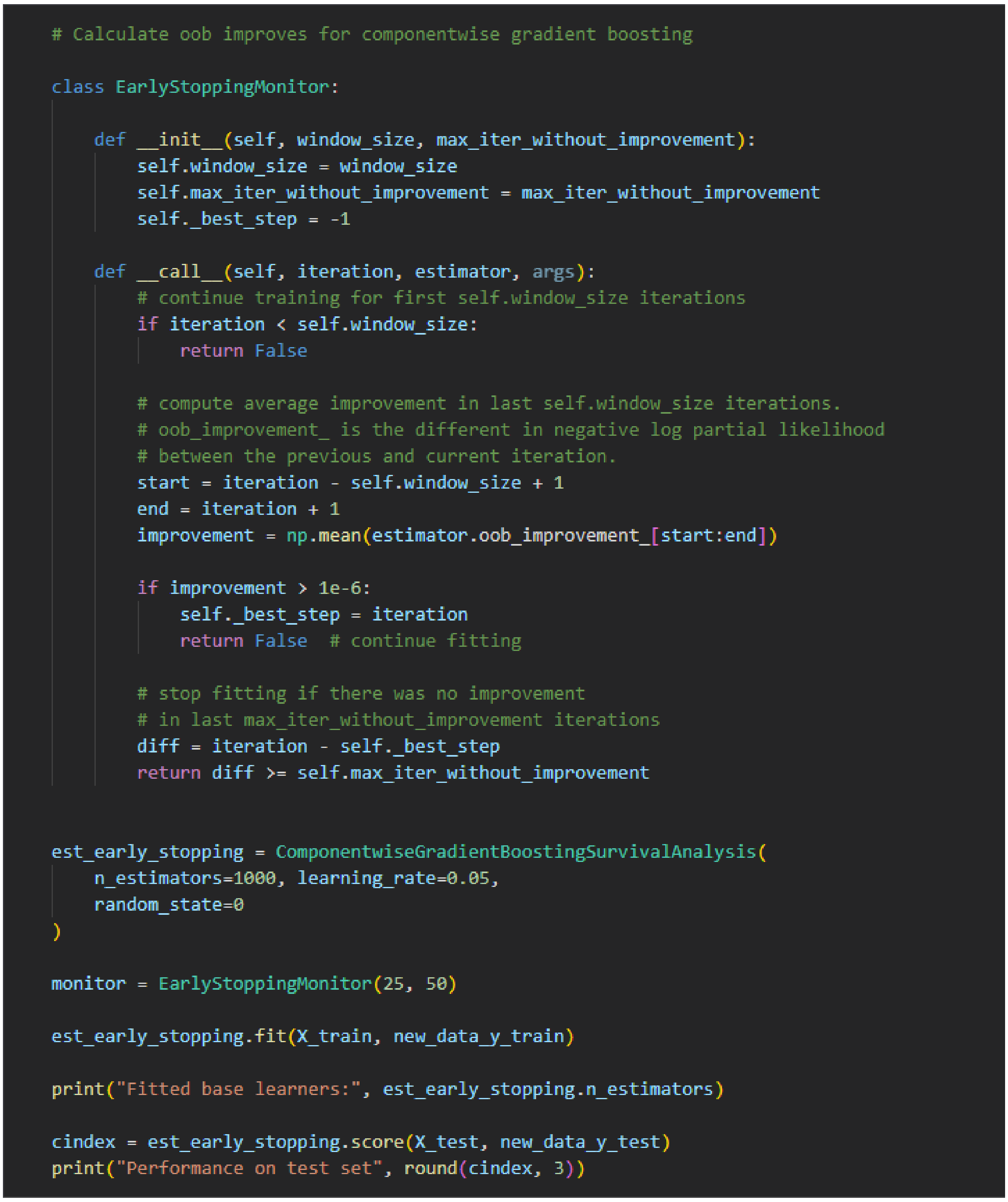
```python
# Plot Parameter tuning for component wise gradient boosting

ax = scores_reg_cw.plot(xlabel="n_estimators", ylabel="concordance index")
ax.grid(True)
```



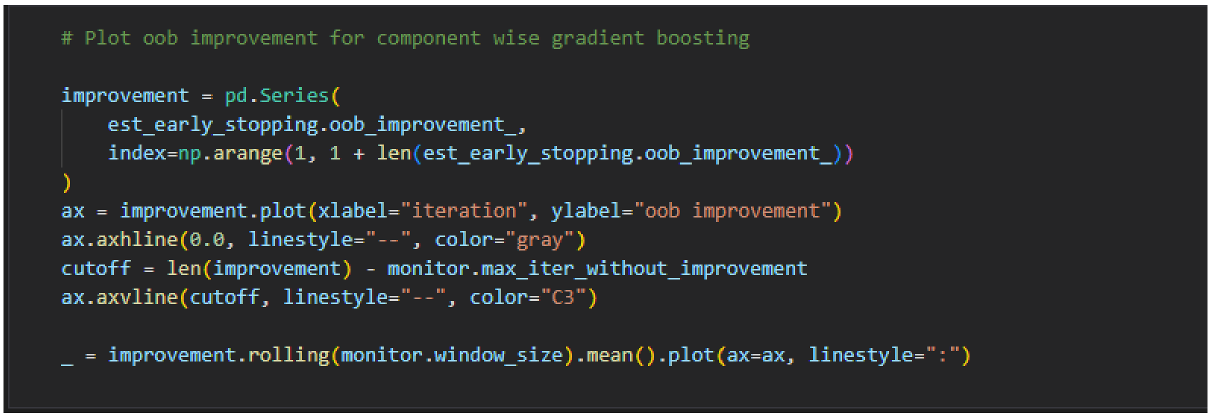
```python
# Calculate oob improves for normal gradient boosting

class EarlyStoppingMonitor:

    def __init__(self, window_size, max_iter_without_improvement):
        self.window_size = window_size
        self.max_iter_without_improvement = max_iter_without_improvement
        self._best_step = -1

    def __call__(self, iteration, estimator, args):
        # continue training for first self.window_size iterations
        if iteration < self.window_size:
            return False

        # compute average improvement in last self.window_size iterations.
        # oob_improvement_ is the different in negative log partial likelihood
        # between the previous and current iteration.
        start = iteration - self.window_size + 1
        end = iteration + 1
        improvement = np.mean(estimator.oob_improvement_[start:end])

        if improvement > 1e-6:
            self._best_step = iteration
            return False  # continue fitting

        # stop fitting if there was no improvement
        # in last max_iter_without_improvement iterations
        diff = iteration - self._best_step
        return diff >= self.max_iter_without_improvement


est_early_stopping = GradientBoostingSurvivalAnalysis(
    n_estimators=1000, learning_rate=0.05, subsample=0.5,
    max_depth=1, random_state=0
)

monitor = EarlyStoppingMonitor(25, 50)

est_early_stopping.fit(X_train, new_data_y_train, monitor=monitor)

print("Fitted base learners:", est_early_stopping.n_estimators_)

cindex = est_early_stopping.score(X_test, new_data_y_test)
print("Performance on test set", round(cindex, 3))
```



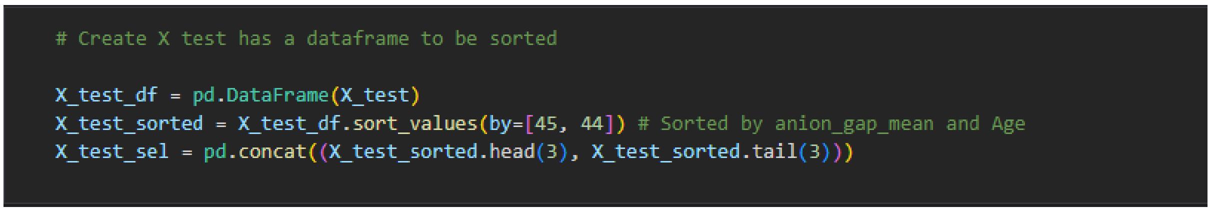
```python
# Plot normal gradient boosting oob improvements

improvement = pd.Series(
    est_early_stopping.oob_improvement_,
    index=np.arange(1, 1 + len(est_early_stopping.oob_improvement_))
)
ax = improvement.plot(xlabel="iteration", ylabel="oob improvement")
ax.axhline(0.0, linestyle="--", color="gray")
cutoff = len(improvement) - monitor.max_iter_without_improvement
ax.axvline(cutoff, linestyle="--", color="C3")

_ = improvement.rolling(monitor.window_size).mean().plot(ax=ax, linestyle=":")
```



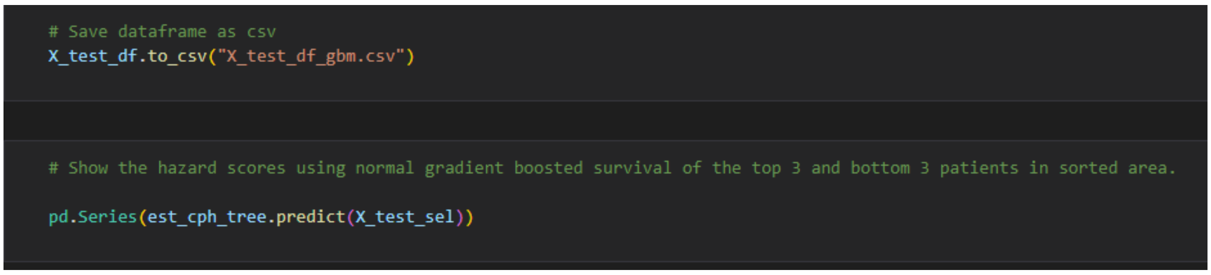
```python
# Calculate oob improves for componentwise gradient boosting

class EarlyStoppingMonitor:

    def __init__(self, window_size, max_iter_without_improvement):
        self.window_size = window_size
        self.max_iter_without_improvement = max_iter_without_improvement
        self._best_step = -1

    def __call__(self, iteration, estimator, args):
        # continue training for first self.window_size iterations
        if iteration < self.window_size:
            return False

        # compute average improvement in last self.window_size iterations.
        # oob_improvement_ is the different in negative log partial likelihood
        # between the previous and current iteration.
        start = iteration - self.window_size + 1
        end = iteration + 1
        improvement = np.mean(estimator.oob_improvement_[start:end])

        if improvement > 1e-6:
            self._best_step = iteration
            return False  # continue fitting

        # stop fitting if there was no improvement
        # in last max_iter_without_improvement iterations
        diff = iteration - self._best_step
        return diff >= self.max_iter_without_improvement


est_early_stopping = ComponentwiseGradientBoostingSurvivalAnalysis(
    n_estimators=1000, learning_rate=0.05,
    random_state=0
)

monitor = EarlyStoppingMonitor(25, 50)

est_early_stopping.fit(X_train, new_data_y_train)

print("Fitted base learners:", est_early_stopping.n_estimators)

cindex = est_early_stopping.score(X_test, new_data_y_test)
print("Performance on test set", round(cindex, 3))
```



```python
# Plot oob improvement for component wise gradient boosting

improvement = pd.Series(
    est_early_stopping.oob_improvement_,
    index=np.arange(1, 1 + len(est_early_stopping.oob_improvement_))
)
ax = improvement.plot(xlabel="iteration", ylabel="oob improvement")
ax.axhline(0.0, linestyle="--", color="gray")
cutoff = len(improvement) - monitor.max_iter_without_improvement
ax.axvline(cutoff, linestyle="--", color="C3")

_ = improvement.rolling(monitor.window_size).mean().plot(ax=ax, linestyle=":")
```

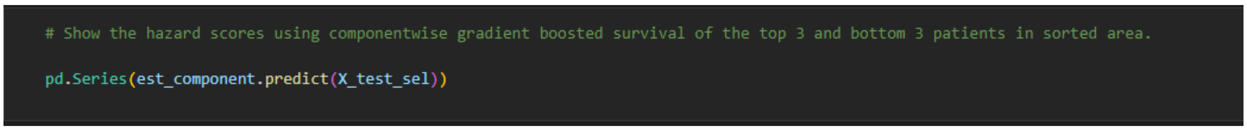


```python
# Create X test has a dataframe to be sorted

X_test_df = pd.DataFrame(X_test)
X_test_sorted = X_test_df.sort_values(by=[45, 44]) # Sorted by anion_gap_mean and Age
X_test_sel = pd.concat((X_test_sorted.head(3), X_test_sorted.tail(3)))
```

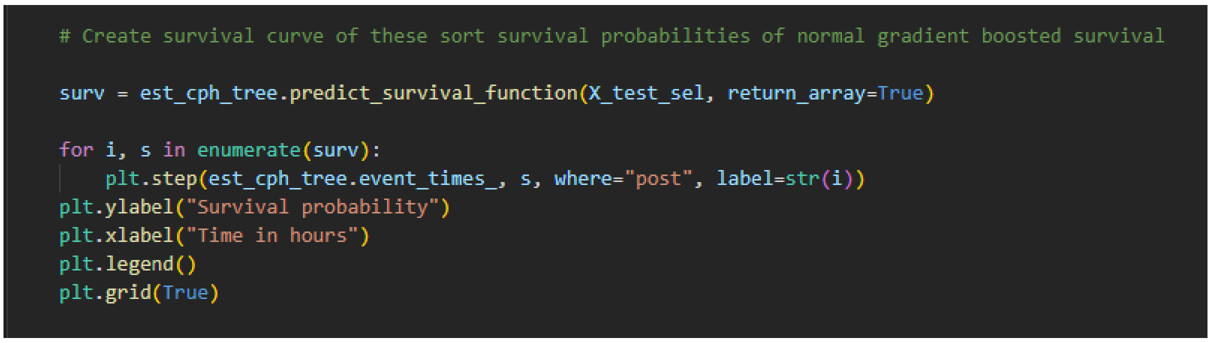


```python
# Save dataframe as csv
X_test_df.to_csv("X_test_df_gbm.csv")
```

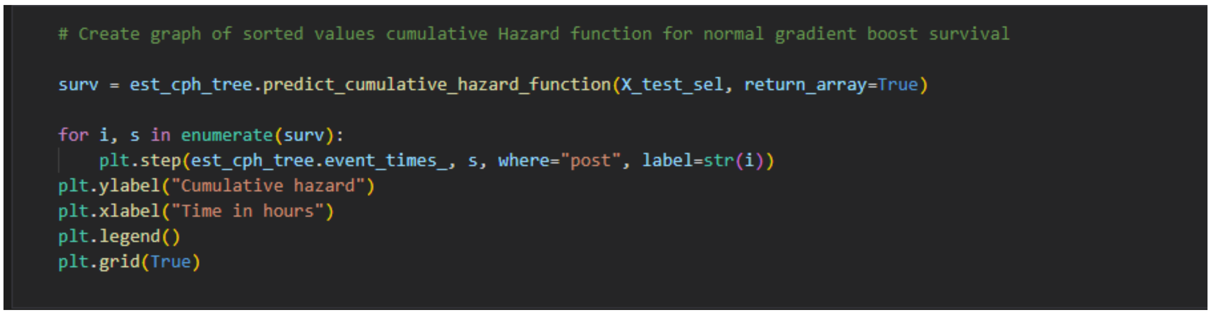
```python
# Show the hazard scores using normal gradient boosted survival of the top 3 and bottom 3 patients in sorted area.

pd.Series(est_cph_tree.predict(X_test_sel))
```



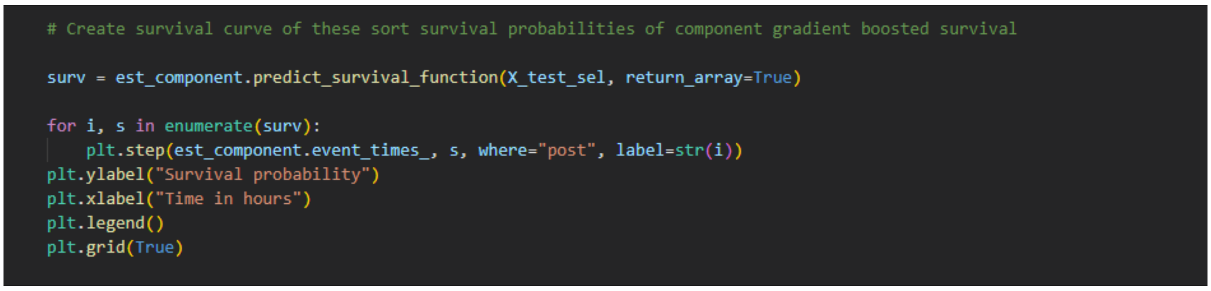
```python
# Show the hazard scores using componentwise gradient boosted survival of the top 3 and bottom 3 patients in sorted area.

pd.Series(est_component.predict(X_test_sel))
```



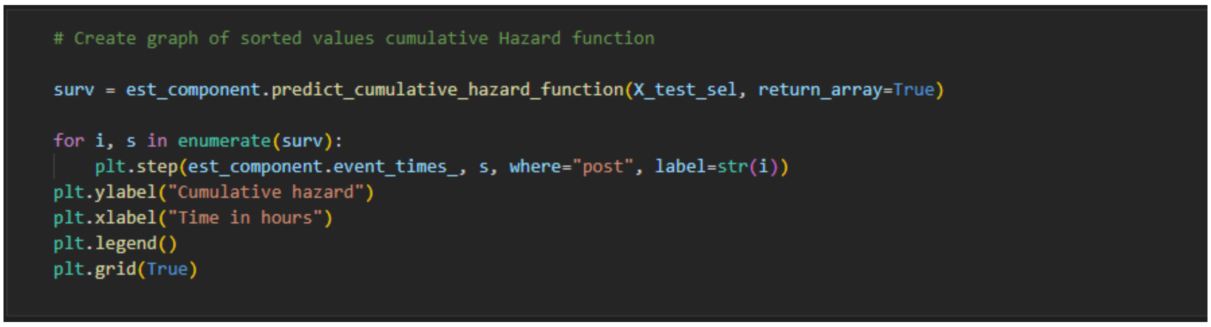
```python
# Create survival curve of these sort survival probabilities of normal gradient boosted survival

surv = est_cph_tree.predict_survival_function(X_test_sel, return_array=True)

for i, s in enumerate(surv):
    plt.step(est_cph_tree.event_times_, s, where="post", label=str(i))
plt.ylabel("Survival probability")
plt.xlabel("Time in hours")
plt.legend()
plt.grid(True)
```



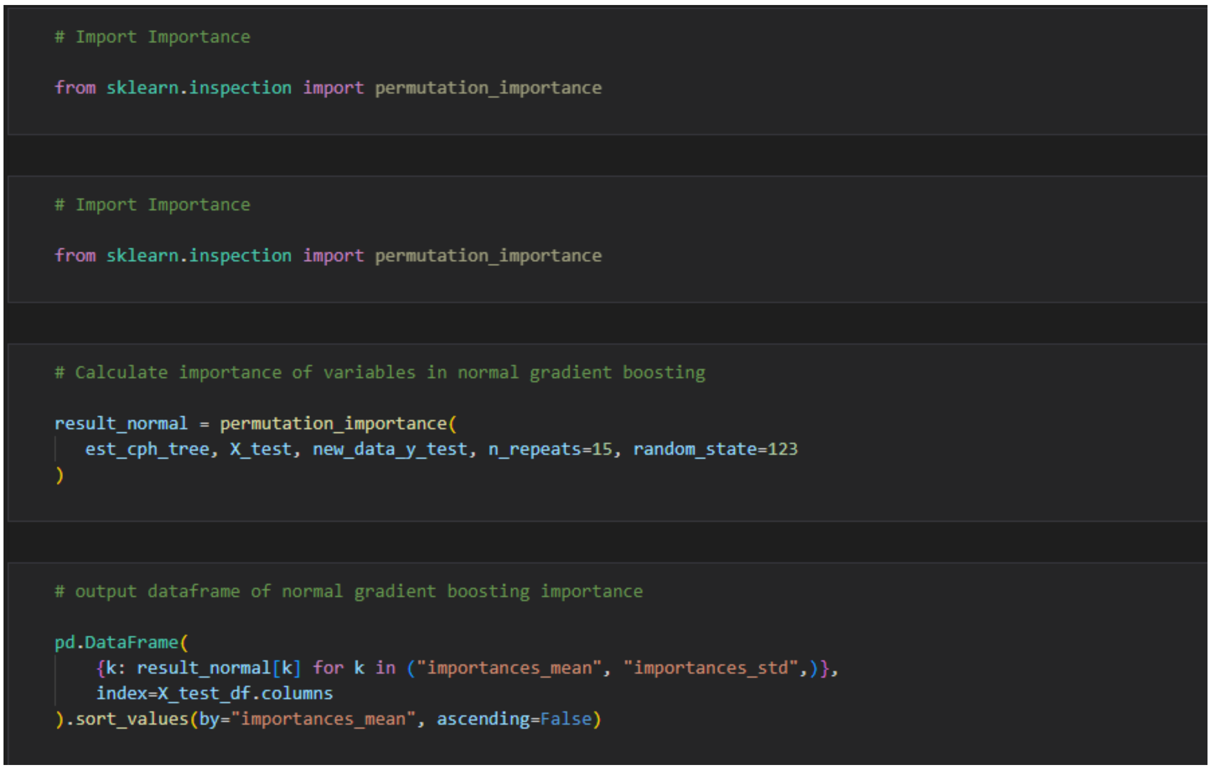
```python
# Create graph of sorted values cumulative Hazard function for normal gradient boost survival

surv = est_cph_tree.predict_cumulative_hazard_function(X_test_sel, return_array=True)

for i, s in enumerate(surv):
    plt.step(est_cph_tree.event_times_, s, where="post", label=str(i))
plt.ylabel("Cumulative hazard")
plt.xlabel("Time in hours")
plt.legend()
plt.grid(True)
```



```python
# Create survival curve of these sort survival probabilities of component gradient boosted survival

surv = est_component.predict_survival_function(X_test_sel, return_array=True)

for i, s in enumerate(surv):
    plt.step(est_component.event_times_, s, where="post", label=str(i))
plt.ylabel("Survival probability")
plt.xlabel("Time in hours")
plt.legend()
plt.grid(True)
```



```python
# Create graph of sorted values cumulative Hazard function

surv = est_component.predict_cumulative_hazard_function(X_test_sel, return_array=True)

for i, s in enumerate(surv):
    plt.step(est_component.event_times_, s, where="post", label=str(i))
plt.ylabel("Cumulative hazard")
plt.xlabel("Time in hours")
plt.legend()
plt.grid(True)
```

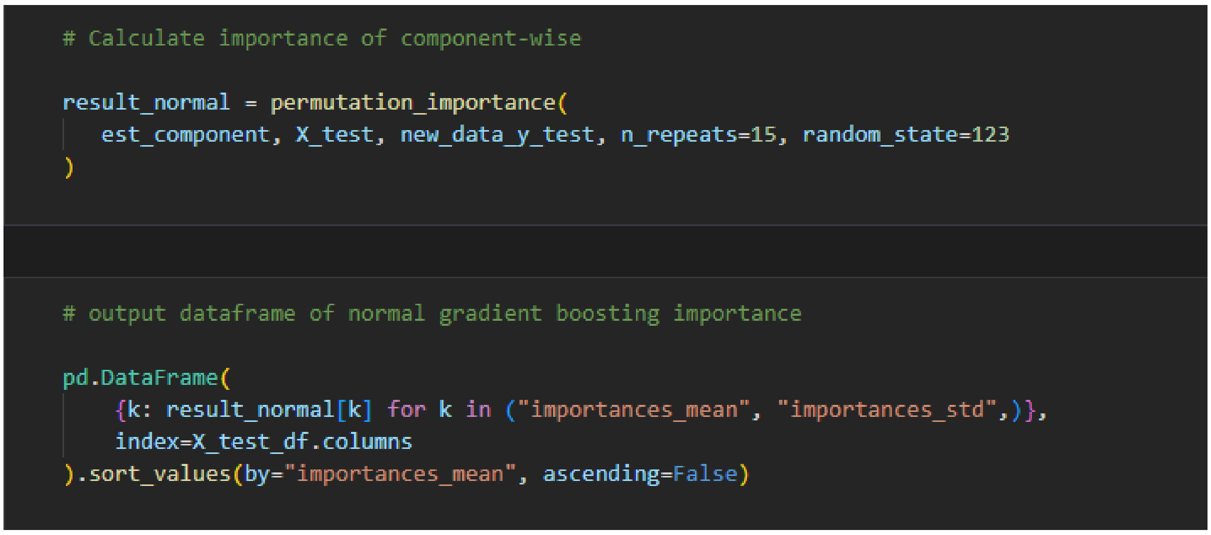


```python
# Import Importance

from sklearn.inspection import permutation_importance
```

```python
# Import Importance

from sklearn.inspection import permutation_importance
```

```python
# Calculate importance of variables in normal gradient boosting

result_normal = permutation_importance(
    est_cph_tree, X_test, new_data_y_test, n_repeats=15, random_state=123
)
```

```python
# output dataframe of normal gradient boosting importance

pd.DataFrame(
    {k: result_normal[k] for k in ("importances_mean", "importances_std",)},
    index=X_test_df.columns
).sort_values(by="importances_mean", ascending=False)
```



```python
# Calculate importance of component-wise

result_normal = permutation_importance(
    est_component, X_test, new_data_y_test, n_repeats=15, random_state=123
)
```

```python
# output dataframe of normal gradient boosting importance

pd.DataFrame(
    {k: result_normal[k] for k in ("importances_mean", "importances_std",)},
    index=X_test_df.columns
).sort_values(by="importances_mean", ascending=False)
```

7 References

McNamara, R. L., Kennedy, K. F., Cohen, D. J., Diercks, D. B., Moscucci, M., Ramee, S., Wang, T. Y., Connolly, T., & Spertus, J. A. (2016). Predicting In-Hospital Mortality in Patients With Acute Myocardial Infarction. *Journal of the American College of Cardiology*, *68*(6), 626–635. https://doi.org/10.1016/j.jacc.2016.05.049

Bewick, V., Cheek, L., & Ball, J. (2004). Statistics review 12: survival analysis. *Critical care (London, England)*, *8*(5), 389–394. https://doi.org/10.1186/cc2955

Lee, D. K. K., Chen, N., & Ishwaran, H. (2021). BOOSTED NONPARAMETRIC HAZARDS WITH TIME-DEPENDENT COVARIATES. *Annals of statistics*, *49*(4), 2101–2128. https://doi.org/10.1214/20-aos2028

Mortazavi, B. J., Desai, N., Zhang, J., Coppi, A., Warner, F., Krumholz, H. M., & Negahban, S. (2017). Prediction of Adverse Events in Patients Undergoing Major Cardiovascular Procedures. *IEEE journal of biomedical and health informatics*, *21*(6), 1719–1729. https://doi.org/10.1109/JBHI.2017.2675340

Ma, J., Lee, D. K. K., Perkins, M. E., Pisani, M. A., & Pinker, E. (2019). Using the Shapes of Clinical Data Trajectories to Predict Mortality in ICUs. *Critical care explorations*, *1*(4), e0010. https://doi.org/10.1097/CCE.0000000000000010

Shirly Wang, Matthew B. A. McDermott, Geeticka Chauhan, Marzyeh Ghassemi, Michael C. Hughes, and Tristan Naumann. 2020. MIMIC-Extract: a data extraction, preprocessing, and representation pipeline for MIMIC-III. In Proceedings of the ACM Conference on Health, Inference, and Learning (CHIL '20). Association for Computing Machinery, New York, NY, USA, 222–235. https://doi.org/10.1145/3368555.3384469

Kuitunen, I., Ponkilainen, V. T., Uimonen, M. M., Eskelinen, A., & Reito, A. (2021). Testing the proportional hazards assumption in cox regression and dealing with possible non-proportionality in total joint arthroplasty research: methodological perspectives and review. *BMC musculoskeletal disorders*, *22*(1), 489. https://doi.org/10.1186/s12891-021-04379-2

Pickett, K. L., Suresh, K., Campbell, K. R., Davis, S., & Juarez-Colunga, E. (2021). Random survival forests for dynamic predictions of a time-to-event outcome using a longitudinal biomarker. *BMC medical research methodology*, *21*(1), 216. https://doi.org/10.1186/s12874-021-01375-x

Hao L, Kim J, Kwon S, Ha ID. Deep Learning-Based Survival Analysis for High-Dimensional Survival Data. *Mathematics*. 2021; 9(11):1244. https://doi.org/10.3390/math9111244

Shah, A. D., Bartlett, J. W., Carpenter, J., Nicholas, O., & Hemingway, H. (2014). Comparison of random forest and parametric imputation models for imputing missing data using MICE: a CALIBER study. *American journal of epidemiology*, *179*(6), 764–774. https://doi.org/10.1093/aje/kwt312

Mogensen, U. B., Ishwaran, H., & Gerds, T. A. (2012). Evaluating Random Forests for Survival Analysis using Prediction Error Curves. *Journal of statistical software*, *50*(11), 1–23. https://doi.org/10.18637/jss.v050.i11

Hong, S., Lynn, H.S. Accuracy of random-forest-based imputation of missing data in the presence of non-normality, non-linearity, and interaction. *BMC Med Res Methodol* **20**, 199 (2020). https://doi.org/10.1186/s12874-020-01080-1

https://scikit-survival.readthedocs.io/en/stable/user_guide/boosting.html

Lu, H., Karimireddy, S.P., Ponomareva, N. &amp; Mirrokni, V.. (2020). Accelerating Gradient Boosting Machines. <i>Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics</i>, in <i>Proceedings of Machine Learning Research</i> 108:516-526 Available from https://proceedings.mlr.press/v108/lu20a.html.